An Adaptive Grid Algorithm for One-Dimensional Nonlinear Equations

Submitted to:

National Aeronautics and Space Administration
Ames Research Center
Moffett Field, California

Submitted by:

William E. Gutierrez
and
Richard G. Hills

Department of Mechanical Engineering
New Mexico State University
Las Cruces, New Mexico 88003

Final Report

Contract # NAG 2-474
Grant # NAG 70070

August 1990

An Adaptive Grid Algorithm for One-Dimensional Nonlinear Equations

Submitted to:

National Aeronautics and Space Administration
Ames Research Center
Moffett Field, California

Submitted by:

William E. Gutierrez
and
Richard G. Hills

Department of Mechanical Engineering
New Mexico State University
Las Cruces, New Mexico 88003

Final Report

Contract # NAG 2-474
Grant # NAG 70070

August 1990

ABSTRACT

Richards' equation, which models the flow of liquid through unsaturated porous media, is highly nonlinear and difficult to solve. Steep gradients in the field variables require the use of fine grids and small time step sizes. The numerical instabilities caused by the nonlinearities often require the use of iterative methods such as Picard or Newton iteration. These difficulties result in large CPU requirements in solving Richards' equation. With this in mind, adaptive and multigrid methods are investigated for use with nonlinear equations such as Richards' equation. Attention is focused on one-dimensional transient problems.

To investigate the use of multigrid and adaptive grid methods, a series of problems are studied. First, a multigrid program is developed and used to solve an ordinary differential equation, demonstrating the efficiency with which low and high frequency errors are smoothed out. The multigrid algorithm and an adaptive grid algorithm is used to solve one-dimensional transient partial differential equations, such as the diffusive and convective-diffusion equations. The performance of these programs are compared to that of the Gauss-Seidel and tridiagonal methods. The adaptive and multigrid schemes outperformed the Gauss-Seidel algorithm, but were not as fast as the tridiagonal method. The adaptive grid scheme solved the problems slightly faster than the multigrid method.

To solve nonlinear problems, Picard iterations are introduced into the adaptive grid and tridiagonal methods. Burgers' equation is used as a test problem for the two algorithms. Both methods obtain solutions of comparable accuracy for similar time increments. For the Burgers' equation, the adaptive grid method finds the solution approximately three times faster than the tridiagonal method. Finally, both schemes are used to solve the water content formulation of the Richards' equation. For this problem, the adaptive grid method obtains a more accurate solution in fewer work units and less computation time than required by the tridiagonal method. The performance of the adaptive grid method tends to degrade as the solution process proceeds in time, but still remains faster than the tridiagonal scheme.

Table of Contents

Page

List of Tables vi

List of Figures vii

Nomenclature x

Chapter 1 - Introduction 1

1.1 Background 1

1.2 Scope 2

Chapter 2 - Literature Review 4

Chapter 3 - Theory 11

3.1 FAS Multigrid Method 12

3.2 Adaptive Grid Method 29

3.3 Transient Algorithms 36

3.4 Nonlinear Adaptive Grid Algorithm 38

Chapter 4 - Test Problems 42

4.1 Test Problem #1 42

4.2 Test Problem #2 44

4.3 Test Problem #3 45

4.4 Test Problem #4 47

4.5 Test Problem #5 49

Chapter 5 - Results 54

5.1 Results for Test Problem #1 55

5.2 Results for Test Problem #2 65

5.3 Results for Test Problem #3 72

5.4 Results for Test Problem #4 77

5.5 Results for Test Problem #5 88

Chapter 6 - Conclusions and Recommendations 107

6.1 Conclusions 107

6.2 Recommendations 111

References 114

List of Tables

Table		Page
5.1.1	Results for Test Problem #1a	56
5.1.2	Results for Test Problem #1b	56
5.1.3	Test Problem #1a, Results	59
5.2.1	Test Problem #2, Multigrid Results	67
5.3.1	Test Problem #3 Results; Time = 0.050	74
5.4.1	Test Problem #4 Results, Variations in *toler*	80
5.4.2a	Test Problem #4 Results, Variations in ϵ	85
5.4.2b	Test Problem #4 Results, More Variations in ϵ	85
5.4.3	Test Problem #4 Results, Time Increment Varied	87
5.5.1	Test Problem #5, Results; 481 Fine Node Grid	91
5.5.2	Test Problem #5, Results; 641 Fine Node Grid	93
5.5.3	Test Problem #5, Results for Time = 5 days	97
5.5.4	Test Problem #5, Results for Time = 15 days	98
5.5.5	Test Problem #5, Results for Time = 35 days	99
5.5.6	Results For the Alternate Tridiagonal Method	106
6.1.1	Summary of Representative Results	110

List of Figures

Figures		Page
3.1.1	Convergence of High and Low Frequency Terms Using the Gauss-Seidel Method	15
3.1.2	Aliasing	17
3.1.3	Linear Interpolation	23
3.1.4	Cubic Interpolation	23
3.1.5	Direct Injection	24
3.1.6	Weighted Average (full weighting)	24
3.1.7	Multigrid Flowchart	26
3.2.1	Adaptive Grid Flowchart	31
3.2.2	Adaptive Grids	35
3.4.1	Cubic Interpolation at a Front	39
3.4.2	Linear Interpolation at a Front	39
3.4.3	Nonlinear Adaptive Grid Algorithm	41
4.5.1	Finite Difference Discretization with Nodes Centered in Grid Volumes	53
5.1.1	Solution to Problem #1a	57
5.1.2	Effects of Varying the Number of Multigrid Levels	57
5.1.3	Solution to Problem #1b, Using Original Switching Parameters	62
5.1.4	Solution to Problem #1b, Using Alternate Switching Parameters	62
5.1.5	Effects of Varying the Switching Parameters	64

5.2.1 Multigrid Solution to Problem #2; Analytical and Multigrid Solutions Overlie 66

5.2.2 Adaptive Grid Solution to Problem #2; Analytical and Multigrid Solutions Overlie 66

5.2.3 Work - Case #1 69

5.2.4 Work - Case #2 69

5.3.1 Adaptive Grid Solution to Problem #3, Snapshot View 73

5.3.2 Adaptive Grid Solution to Problem #3, Time Histories; Analytical and Multigrid Solutions Overlie 73

5.3.3 Multigrid Solution to Problem #3, Snapshot View 76

5.3.4 Multigrid Solution to Problem #3, Time Histories; Analytical and Multigrid Solutions Overlie 76

5.4.1 Adaptive Grid Solution to Problem #4, Burgers' Equation . . . 79

5.4.2 Work vs. Log(ϵ), Burgers' Equation 83

5.4.3 CPU Time vs. Log(ϵ), Burgers' Equation 83

5.4.4 Effects on the Solution Caused by Varying ϵ 84

5.4.5 Diffusion of Front Due to Large Time Increments 84

5.4.6 CPU Time vs. Time Increment, Burgers' Equation 86

5.5.1 Adaptive Grid vs. Tridiagonal Solution to Problem #5, Richards' Equation 89

5.5.2 Adaptive Grids Used at Time = 15 Days 95

5.5.3 Adaptive Grid Residual Error, Case 1 100

5.5.4 Adaptive Grid Residual Error, Case 2 100

5.5.5 Adaptive Grid Residual Error, Case 3 101

5.5.6 CPU Time vs. Time Step; Richards' Equation at Time = 5 days 104

5.5.7 CPU Time vs. Time Step; Richards' Equation at Time = 15 days 104

5.5.8 CPU Time vs. Time Step; Richards' Equation at Time = 35 days 105

Nomenclature

English

C - coefficient relating volumetric water content and tension

e - norm of the residual function

e^k - residual norm on grid level k

e_{AG} - residual norm obtained from the adaptive grid method

$\overline{e}, \overline{e}^k$ - norm of the residuals prior to last relaxation

$F, f(x)$ - original forcing function (right hand side)

F^H - discrete approximation of the original forcing function on a coarse grid H

F^h - discrete approximation of the original forcing function on a fine grid h

f^k - discrete forcing function on grid level k

G^H - denotes a coarse, uniformly spaced grid

G^h - denotes a fine, uniformly spaced grid

H - uniform spacing between nodes on a coarse grid

h - uniform spacing between nodes on a fine grid

I_k^{k+1} - linear interpolation from grid level k to $k+1$

I_{k+1}^k - interpolation (restriction) from grid level $k+1$ to k

$I\!I_k^{k+1}$ - cubic interpolation to the next finer grid

i - positive integer denoting a specific node

j - positive integer used in identifying time steps

$\hat{j}$ - number of time increments used for 'local time refinements'

K - hydraulic conductivity

K_s - saturated hydraulic conductivity of soil

k - positive integer used to identify a grid level

L - differential operator

L^H - discrete differential operator on a coarse grid H

L^h - discrete differential operator on a fine grid h

$\mathcal{L}$ - size of the domain

l - finest grid level that has been visited by the algorithm

m - positive integer denoting the finest grid level

$\hat{m}$ - order of differential equations being solved

$\tilde{m}$ - model parameter used with problem #5

n - number of nodes on the grid of interest;

$\tilde{n}$ - model parameter used with problem #5

p - order of the approximation scheme

$\hat{p}$ - mesh size ratio between two adjacent grid levels

q - flux boundary condition

r - number of relaxation sweeps on each finer level per multigrid cycle

r^h - residual function (vector of residual values)

r^k - residual function on grid level k

$\overline{r}^k$ - residual function on grid level k prior to the last relaxation sweep

S_e - water retention curve

t - time

toler - convergence criteria used in the nonlinear adaptive grid algorithm to determine whether more Picard iterations are needed

tolerce - tolerance value used with the nonlinear tridiagonal method to detect a premature convergence of the residual onto a value greater than that desired

Δt - time increment

$U, u(x)$ - true solution

U^H - discrete approximation of the true solution on a coarse grid H

U^h - discrete approximation of the true solution on a fine grid h

u - field variable for problems #1 through #4

u^H - current approximate of the solution on a coarse grid H

u^h - current approximate of the solution on a fine grid h

$\overline{u}^H, \overline{u}^h$ - approximate solution prior to the previous relaxation

v - error present in the approximate solution u

$\overline{v}$ - errors present prior to the last relaxation

x - spatial dimension

Δx - distance between nodes

Δx_{k-1} - distance between nodes on grid level $k-1$

Greek

α - multigrid 'switching parameter' used to determine the convergence criteria for the finest level visited

$\tilde{\alpha}$ - model parameter used in problem #5

δ - multigrid 'switching parameter' governing the convergence criteria on coarser grid which have been previously visited

ϵ, ϵ^k - convergence criteria

η - 'stopping factor' used to detect an unacceptable slowing of the convergence rate

θ - volumetric water content

θ_r - residual water content

μ - convergence rate

$\overline{\mu}$ - smoothing factor

$\mu(\vartheta)$ - spectral convergence rate

θ_s - saturated water content

τ^H, τ^h - estimate of the local truncation error

τ_h^H - relative (local) truncation error

τ^k - current estimate of the truncation error

τ_i^{k-1} - relative truncation error on level $k-1$, node i

Chapter 1

Introduction

1.1 Background

Many partial differential equations exist which are very difficult to solve analytically. The solution to such problems can often be found by numerical methods, such as through the use of finite differences. The use of finite differences leads to systems of linear or nonlinear algebraic equations. Solving the resulting system of equations by conventional relaxation methods is CPU intensive, especially for large systems. The application of the multi-level technique to conventional relaxation methods accelerates the rate at which they converge to the solution. As a result, the multigrid technique has found widespread use for the solution of multi-dimensional partial differential equations which arise in computational fluid dynamics and physics. A feature of the multigrid technique is the ease by which it can be modified to handle adaptive grids, which are ideally suited for problems with steep local gradients in the state variables.

Like multigrid methods, adaptive grid algorithms also use a sequence of finer and finer uniform grids to solve a problem, but, may restrict the finer grids to small portions of the domain. This method seeks to identify those portions of the domain requiring the use of the finer grids in order to achieve

a desired accuracy, and then solves the problem within these regions using the finer grids. Such regions often occur in problems which exhibit steep gradients in the solution [Hedstrom and Rodrique, 1982]. Thus, the adaptive grid method appears to be well suited toward finding the solution to problems possessing these steep local gradients.

1.2 Scope

The aim of this work is to first investigate the use of the multigrid and adaptive grid techniques [Brandt, 1977] for finding the solution to one-dimensional, differential equations which may contain steep local gradients. Then, the most promising method is adapted to find the solution to two one-dimensional nonlinear problems. The nonlinear problems are the viscous Burgers' equation, which contains a moving front with steep gradients in the solution; and, the water content formulation of the Richards' equation modelling one-dimensional liquid flow in unsaturated soils. The Richards' equation is highly nonlinear and also possesses solutions which contain moving fronts with very steep gradients.

The application of the multigrid algorithm is presented first for a one-dimensional steady state diffusion problem. It is then expanded to include one-dimensional transient problems. The time-dependent problems are also solved using the adaptive grid method. The adaptive grid technique is then applied toward solving the nonlinear Burgers' equation.

The resulting algorithms are verified by comparing their results to those obtained using other numerical methods. Where possible, the numerical solutions are also verified by comparison to the analytical solutions. The efficiency of the algorithms are determined by comparing the CPU time needed to solve the problem to that needed for a conventional relaxation method and direct solver.

Finally, the adaptive grid method is applied to the nonlinear one-dimensional water content formulation of the Richards equation. The adaptive grid method has not previously been applied to this problem. The adaptive grid solution is compared to that obtained using a tridiagonal method employing Picard iterations.

Chapter 2

Literature Review

Usually, when solving a Partial Differential Equation (PDE) by numerical means, one first discretizes the problem onto a fine grid. This often results in a very large system of simultaneous algebraic equations. The resulting system of equations is then solved using an iterative solver. Such a method is easy to implement into a computer program, but iterative solvers used in this manner do have some problems. One such problem is the slow convergence of the solution due to inefficiencies in resolving low frequency errors (errors with a long wavelength). Another is the lack of interplay between the discretization and the solution processes which overlooks useful information [Brandt, 1977]. By performing a Fourier analysis of the problem on a fine grid, it becomes evident that high frequency errors are quickly resolved in just a few relaxation sweeps and that the low frequency errors converge very slowly. After three relaxations the high frequency error terms are reduced by almost an order of magnitude [Brandt, 1977, 1979]. This leads to the multigrid method, which discretizes the problem onto a sequence of coarser and coarser grids. The multigrid method is a general technique by which the performance (in terms of CPU time) of an iterative (conventional) solver is improved. Multigrid techniques normally solve an elliptic PDE in O(n) operations whereas

iterative solvers such as Gauss-Seidel require $O(n^2)$ operations [Brandt, 1977; Wesseling, 1982].

The basic multigrid method is a systematic scheme uniting the solution process on several grid levels by combining relaxation sweeps with corrections involving the current estimates of the solution on both the fine and coarse grids. The coarse grids are used to provide the finer grids with an estimate of the solution containing the low frequency information, which is difficult (computationally expensive) to obtain on the fine grids. The fine grids are used to resolve the high frequency errors in the solution, which are then used to improve the coarse grid solution. Thus, the interactions between the fine and coarse grids serve to improve the performance of an iterative solver.

One basic multigrid method is named the Coarse Grid Correction (CGC) scheme. This scheme begins by solving the problem of interest only on the finest grid, with the coarser grids used to solve residual equations. The coarse grid solution to the residual equations is then used as a correction to the solution on the finer grids [Alcouffe et al., 1981; Brandt, 1977; Jesperson, 1984]. The CGC method is primarily for linear problems and cannot be applied to composite (or adaptive) grids. Extending CGC for use with nonlinear problems is "messy" [Brandt, 1979, 1982].

Another class of multigrid methods are the Full Multigrid (FMG) algorithms. These methods discretize and solve the problem of interest onto all

grid levels. They begin the solution process on the coarsest grid by obtaining the first approximation to the solution. The full multigrid methods have some very attractive features as they can solve a problem to the level of truncation errors and tend to be very forgiving of small mistakes. For example, conceptual or programming errors often just slow down the rate of convergence and have no effect on the actual solution [Brandt, 1982].

One such commonly used method is the Full Approximation Scheme (FAS) which is also referred to as the Full Approximation Storage method. This scheme is similar to CGC except that the problem of interest is solved on all grid levels. As before, the coarse grids are used to correct the solution on the finer grids. FAS also differs in that the fine grid solution is used to modify the coarse grid forcing function so as to coincide with the fine grid solution. The full approximation scheme has several advantages and provides a general algorithm for both linear and nonlinear problems. When used to solve nonlinear problems, global linearization is not required. The only linearization needed is the local linearization employed by the relaxation routine. FAS can also handle "accommodative" grids (adaptive and composite grids) and is easily modified to do so. It also gives a good estimate of the truncation error which is useful in defining stopping and adaptive grid criteria [Brandt, 1977, 1979, 1982].

Adaptive and composite grids are used to reduce the computation time

required to solve certain problems and lead to non-uniform grids on a global scale. "Non-uniform resolution is needed in many, perhaps most, practical problems" [Brandt, 1982]. Both composite and adaptive grids may be viewed as a series of uniform grids which do not necessarily extend over the entire domain. The finer grids are needed near singularities, non-smooth boundaries, wave fronts, shocks, etc.

One can define composite grids as a union of uniform subgrids, usually positioned such that every node on the coarse grid corresponds to a node (usually every other node) on the next finer grid. It is important to note that these subgrids do not have to cover the entire domain, and that a subgrid may extend over a portion of the domain covered by another subgrid. Subgrids in different subdomains may also have different levels of refinement (node spacing). This method is flexible in that local grid refinement is done by extending subregions. Use of composite grid levels as a multigrid sequence yields an efficient solution process [Brandt, 1979]. One problem with this approach is that the composite grids need to be constructed prior to beginning the solution process. Since these grids remain fixed throughout the solution process, a-priori knowledge of the problem and solution is required to adequately form the composite grids.

Adaptive grids are similar to composite grids which use a specified number of uniform grids. The main difference is that with adaptive grids, an open

ended sequence of uniform grids is employed in order to find the solution. This allows for the addition of finer grids (or subgrids) as needed. The subgrids are again constructed as a sequence of uniform grids (much like that used with multigrid). The finer grids are used in those portions of the domain requiring additional refinement. These regions are identified by comparing the local truncation errors (as estimated by the program) to some desired convergence criteria. Using the adaptive grid levels as a multigrid sequence, provides an efficient solution process with relaxation taking place only on uniform grids and local uniform subgrids. Since the FAS scheme combines the full solution (and not just a correction term) on all grid levels and provides a good estimate of the truncation error, it is especially well suited for use with adaptive grids, resulting in a "nearly optimal discretization scheme" [Brandt, 1973, 1977, 1979, 1982].

A multigrid method with adaptive grids applied to a one-dimensional problem is presented in Brandt, 1973. Adaptive multigrid algorithms combine the advantages of both methods. The only apparent disadvantage is the more complex programming involved in incorporating adaptive grids into the multigrid technique.

The multigrid process is useful for solving time-dependent problems. An easily implemented transient method is given in the paper by H. Lee and R. Meyers, 1980. In their paper, a multigrid scheme similar to the FAS method

(presented in Brandt, 1977) is extended to include transient partial differential equations. With this scheme, the spatial terms are discretized onto a series of coarser and coarser uniform grids, while the transient terms are discretized using backward finite difference and a fixed time increment. The multigrid method is then used to solve the problem (at each time step) one time step at a time.

A similar approach (using a fixed time step) for composite grids is presented by Heroux and Thomas, 1989. An extension of this method is to use local time stepping. The local time refinements are added in those subregions covered by fine grids. If the coarse grid time step is Δt, then the fine grid time step may be defined as $\Delta t_{fine} = \Delta t/\hat{j}$, where $\hat{j}$ is a positive integer denoting the number of fine grid time steps that the coarse grid time increment is broken up into [Heroux and Thomas, 1989].

A one-dimensional adaptive grid method using local time stepping is given by Hedstrom and Rodrique, 1982. The algorithm presented is recursive and thus may have several levels of refinement in the time domain. In addition, at any time level there may be many fine grids. An important advantage of using local time refinements with the adaptive grid method is that the resulting algorithm can simultaneously track a number of wave fronts.

The multigrid method is used in solving the convection-diffusion equation with strong convective effects by G. F. Carey and Pandanami, 1989. For these

problems, the cell Peclet (or Reynolds) number restricts the size of the node spacing on the coarse grids. If this condition is violated more than once, the multigrid method fails. Also, if the mesh is too coarse, Jacobi and Gauss-Seidel relaxation methods will diverge. The usual way to get around these problems is to use an upwinding scheme or to add artificial dissipation. Additionally, two alternative approaches to overcome these problems, a fine to coarse grid condensation and a local elliptic projection method, are presented by Carey and A. Pandanami, 1989.

Chapter 3

Theory

The FAS multigrid technique is presented in the first part of this chapter. A finite difference formulation of a linear problem on a uniformly spaced grid is considered in order to show how an iterative solver may benefit from the use of coarser grids approximating the same problem. With this in mind, the convergence rate is introduced and used along with local mode analysis to examine the effects of the solver and various grids on the errors in the approximated solution. Next, the criteria for switching from one grid level to another is discussed, followed by the interpolations used when transferring information between levels. The coarse to fine grid correction and a fine to coarse grid correction are presented along with a summary and flowchart of the multigrid method.

Then, the adaptive grid method is introduced. Like the multigrid scheme, the adaptive grid algorithm uses several grid levels to solve a problem. This method solves the problem over the entire domain on a coarse grid with the finer grids being confined to those portions of the domain which have not satisfactorily converged on the coarser grids. The convergence criteria, which is compared to the truncation error, is then discussed along with the method used to estimate the truncation error. This is followed by a presentation on the

use of truncation error to identify the portions of the domain requiring further refinement, and the construction of the adaptive subgrids. Also included is a flowchart of the adaptive grid method.

The final portion of this chapter discusses the application of the algorithms to transient problems. The discretization of the time domain is addressed, followed by a description of how the solution for the new time step is estimated.

3.1 FAS Multigrid Method

Suppose we have a problem of the form

$$LU(x) = F(x), \tag{3.1.1}$$

where L is a linear differential operator, $U(x)$ is the true solution, and $F(x)$ is the forcing function or right hand side. Discretizing the problem using finite differences on a uniformly spaced grid G^h gives

$$L^h U^h = F^h, \tag{3.1.2}$$

where U^h is the discrete approximation of the true solution and h is the uniform spacing between the nodes. Boundary conditions are also discretized and included into the finite difference equations. Equation (3.1.2) is a set of simultaneous algebraic equations and can be solved by use of an iterative or direct solution method. Direct solution methods are not usually used because they are generally more complex and slower than other methods, such as

iterative schemes. An estimate of the solution is required in order for an iterative method to work. How fast the iterative method converges is partially dependant upon the accuracy of the estimate provided. The better the initial estimate is, the faster the method converges. A good initial approximation may be obtained by solving the same problem using a coarser grid in which the nodal spacing (Δx) is larger. This results in a set of simultaneous algebraic equations

$$L^H u^H = F^H \quad \text{on grid} \quad G^H \tag{3.1.3}$$

which is smaller than the previous set. The superscript H denotes a coarser grid (larger grid spacing but a smaller number of unknowns) than that denoted by h. Usually the spacing between nodes on the coarser grid is twice that of the next finer grid. As before, this problem can be solved using an iterative solver. An initial estimate of the solution is again required. For the problem on G^H a still coarser grid can be utilized in a similar manner, and so forth [Brandt, 1977]. Therefore, the coarser grids are used as a means of providing a better estimate of the solution to the finer grids in order to speed up the convergence.

The convergence rate μ is the rate at which the errors in the state variable are reduced per iteration. Defining the estimate to the true solution on G^h as u^h, the errors before ($\overline{v}$) and after (v) an iteration can be written as

$$\overline{v} = U^h - \overline{u}^h \qquad \text{and} \qquad v = U^h - u^h. \tag{3.1.4a,b}$$

The convergence rate may be expressed as

$$\mu = \frac{\|v\|}{\|\bar{v}\|} \tag{3.1.5}$$

and considered to be the factor by which the errors are reduced per iteration sweep. After a few iterations the convergence rate approaches $\mu = 1 - O(h^2)$, which is a very slow rate. So, to reduce the errors an order of magnitude, $O(h^{-2})$ relaxation sweeps must be made [Brandt, 1977]. This difficulty in reducing the low frequency errors is why many iterative solvers are often slow to converge.

To illustrate this effect, the Gauss-Seidel method was used to solve the following problem containing both high and low frequency (short and long wavelength) terms using a grid consisting of 129 nodes.

$$\frac{d^2u}{dx^2} = -\pi^2 \sin(\pi x) - \left(\frac{26\pi}{10}\right)^2 \sin(26\pi x), \quad (0 \le x \le 1), \quad u \equiv u(x) \tag{3.1.6}$$

$$\text{with the boundary conditions} \quad u(0) = u(1) = 0 \ .$$

The analytical solution to (3.1.6) is

$$u(x) = \sin(\pi x) + 0.1 \sin(26\pi x) \ . \tag{3.1.7}$$

As is shown in Figure 3.1.1, the high frequency terms are quickly resolved (within about 10 iterations) while the low frequency terms take a very long time (22,541 iterations) to converge.

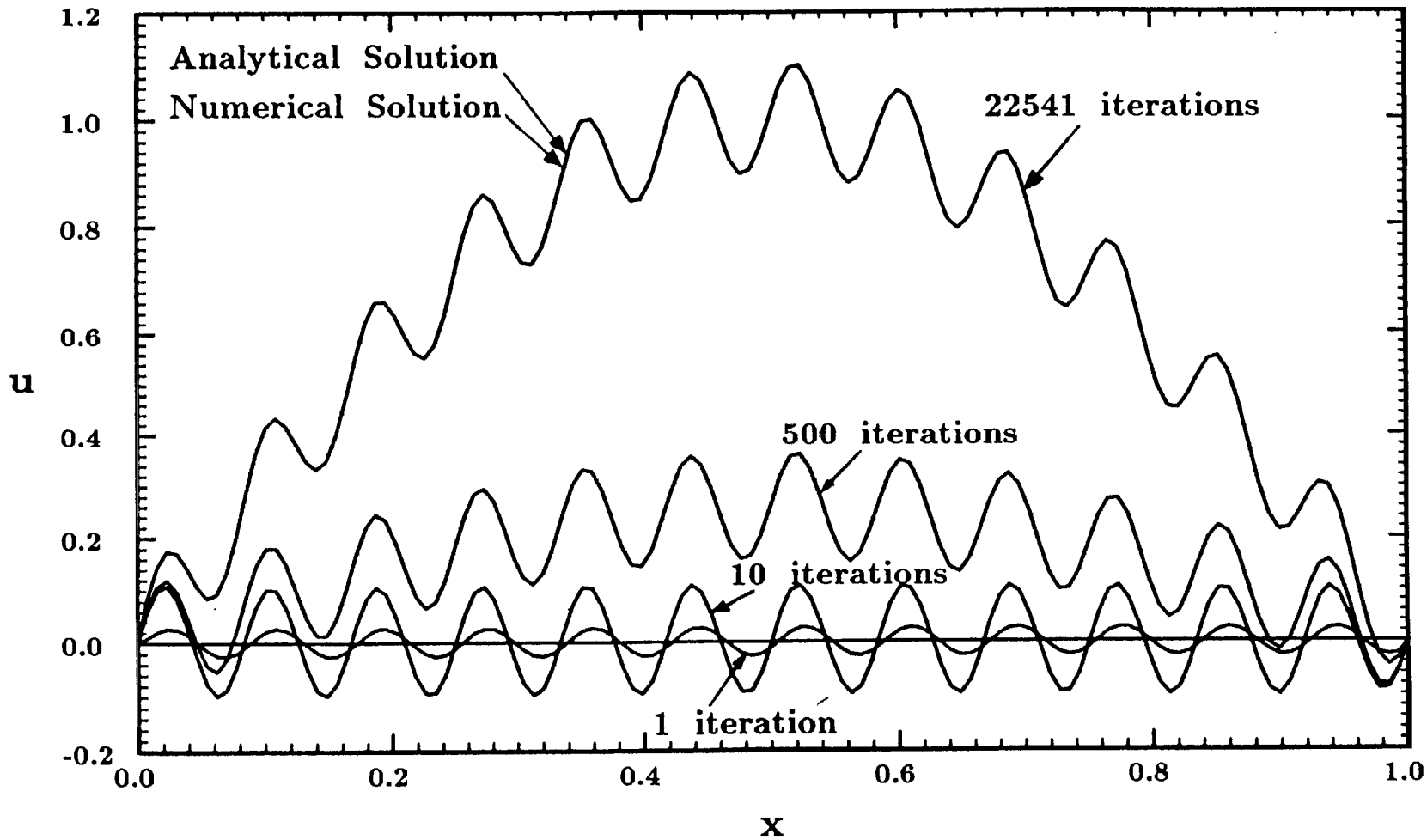

Figure 3.1.1: Convergence of High and Low Frequency Terms

Using the Gauss-Seidel Method

From this viewpoint, the goal of the multigrid technique is to reduce the lower frequency errors on the coarser grids while reducing the higher frequency errors on the finer grids. The convergence rate of the higher frequency error terms can be found by expanding the error into its Fourier components.

The error terms v contain only those errors that are visible on the current grid. Those error components whose frequencies are too high to be resolved on the grid being used appear as low frequency errors. In Figure 3.1.2, a high frequency curve is presented with the grid nodes identified. As the grid is too coarse (the nodes are too far apart) to show the true shape of the curve, the actual curve as viewed from the grid appears to be of a lower frequency than what it actually is. This effect is referred to as aliasing.

The smoothing factor $\overline{\mu}$ is the worst rate of convergence for the high frequency errors visible on the current grid level [Brandt, 1982; Jesperson, 1984]. The smoothing factor is given by

$$\overline{\mu} = \max_{\hat{p}\pi \leq \vartheta \leq \pi} \mu(\vartheta) \ , \quad \text{where} \quad \hat{p} = \frac{h^k}{h^{k-1}} \ , \tag{3.1.8}$$

and $\mu(\vartheta)$ is the spectral convergence rate [Brandt, 1977]. The superscript k refers to a fine grid, while the subscript $k-1$ denotes the next coarser grid level. The relation $\hat{p}$ is the "mesh size ratio" and usually is about 2:1 . The 2:1 ratio should always be used, as it is nearly optimal and is the most convenient and economical ratio for use in the interpolation process [Brandt, 1977]. The smoothing factor can be found by using local mode analysis on the Fourier

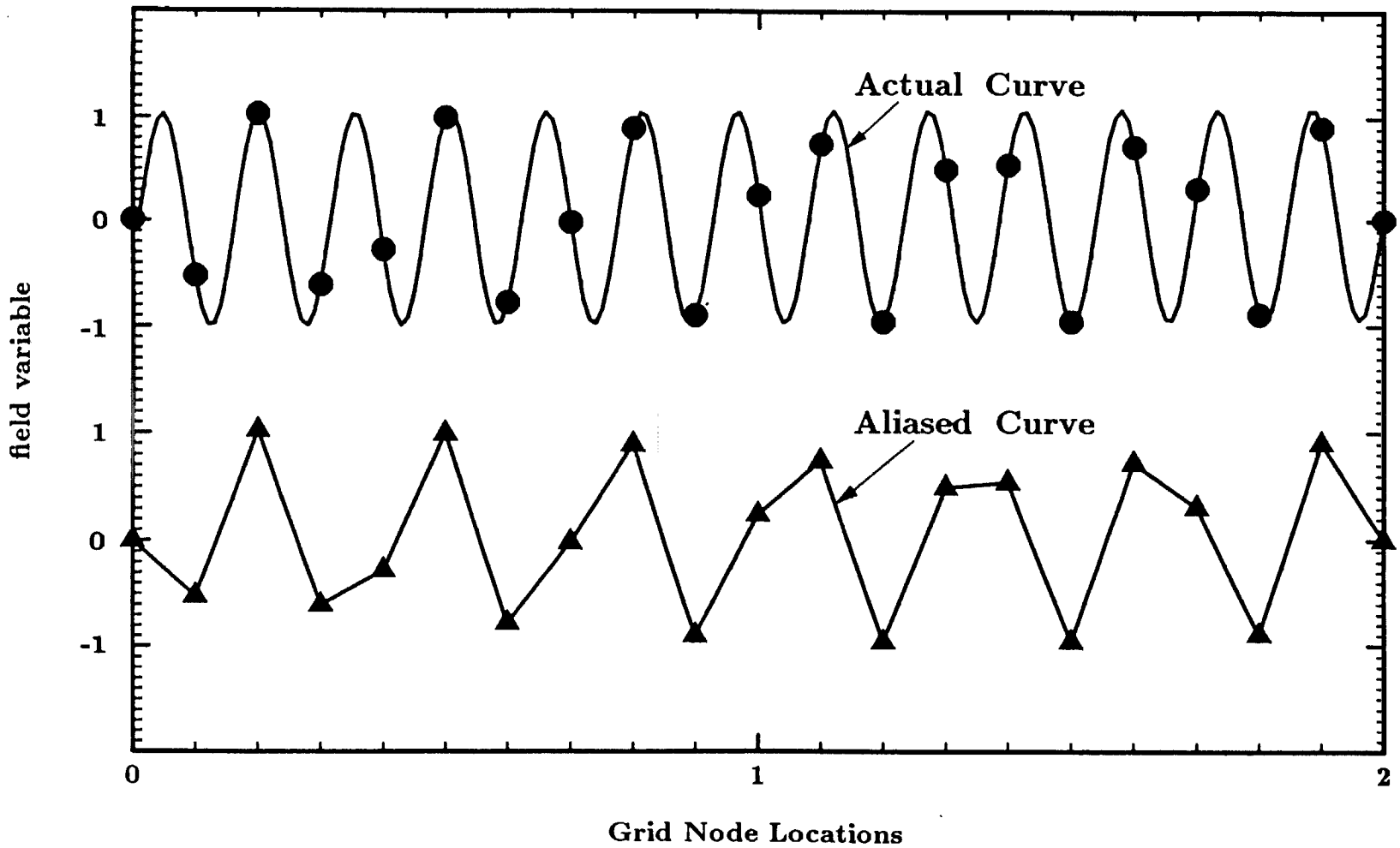

Figure 3.1.2: Aliasing

components of the error. From the smoothing factor "one can explicitly calculate the smoothing rate $|\log \overline{\mu}|^{-1}$ for any given difference equation with any given relaxation scheme" [Brandt, 1977]. The smoothing rate is the number of relaxation sweeps needed to reduce the high frequency errors an order of magnitude. For relaxation methods, the smoothing of high frequency terms can be very reasonable. For example, a Gauss-Seidel sweep over Poisson's equation gives the smoothing factor $\overline{\mu} = 0.45$ and thus, a smoothing rate of 2.86 . This implies that the high frequency errors are reduced by an order of magnitude after about 3 iterations.

In general, when further relaxations at the current level lose their effectiveness, execution of a multigrid algorithm transfers to another (either finer or coarser) level of discretization. A set of criteria is needed to detect when to change over to another grid and to determine whether that grid needs to be a finer or a coarser one. This criteria is partially based on residuals which is a measure of the errors present in the estimate of the solution. By rewriting (3.1.4b) as

$$U^h = u^h + v^h \ , \tag{3.1.9}$$

and then introducing (3.1.9) into (3.1.2) and rearranging terms gives the "residual equation,"

$$r^h = F^h - L^h u^h = L^h v^h \ . \tag{3.1.10}$$

The residual function r^h is a vector containing several values. It would be

much easier to work with a single scalar value, which is a measure of the residuals. This is achieved by taking the norm of the residual function,

$$e^k = \|r^k\| \ , \tag{3.1.11}$$

where k is an integer value denoting the current discretization level.

To check for convergence at the current level, the measure of the residuals e^k is compared to a tolerance ϵ^k. The tolerance ϵ^k is designed such that $e^k < \epsilon^k$ signals convergence [Brandt, 1979]. When $e^k < \epsilon^k$, the problem has converged at the current level and a switch to the next finer grid is made. If the current level happens to be the finest level, then the multigrid procedure is halted as the desired solution has been found.

If the tolerance criteria is not met, the decision to either go to a coarser level or remain at the current one needs to be made. If the convergence rate on the current grid level is slow (high frequency errors visible on this level have been smoothed), a switch to the next coarsest grid is made. The slowing of the convergence rate is detected by comparing the reduction in the residuals to a "stopping factor" η as shown in the equation below:

$$\frac{e}{\bar{e}} \geq \eta \ , \tag{3.1.12}$$

where $\bar{e}$ is the norm of the residual at the previous iteration. As long as (3.1.12) is not satisfied, further efficient error reduction is achieved by additional relaxation sweeps at the current discretization level. When the inequality is met,

relaxations on the next coarser grid become more effective and the algorithm switches to the coarser grid. An appropriate value for η may be taken as an estimate of the smoothing factor μ or found by trial and error. Good relaxation methods have a smoothing factor of about 0.5 [Brandt, 1979]. Increasing the value of η delays the transfer of execution to a coarser grid in favor of continued relaxations on the current level. "Generally the overall multi-grid convergence rate is not very sensitive to increasing η" [Brandt, 1977].

A point to remember is that on the coarsest grid, the problem is solved to the given tolerance ϵ even though the convergence rate may slow down, and thus, require several (less efficient) relaxations at this level. This extra computational work is inexpensive as the coarsest grid contains relatively few unknowns, especially when compared to the finest grid. The use of a direct solver (which solves the problem to the level of truncation error) may be used in place of the relaxation method for the coarsest level to eliminate unwanted relaxation sweeps.

In changing levels, information about the solution must be transferred to another grid. This is done by some type of interpolation process which depends upon whether the destination is a coarser or finer grid. The interpolation process is designated by the operator I_k^{k+1}, where the subscript denotes the current grid level and the superscript denotes the grid level onto which the interpolation is being made. Interpolating from a coarse grid to a fine grid

is referred to as prolongation; while interpolation from a fine grid to a coarse grid is called restriction.

When interpolating to a finer grid, the order of the interpolations must be at least equal to the order of the differential equations ($\hat{m}$) being solved [Brandt, 1977, 1979, 1982]. Using lower order interpolations may result in the creation of significant high frequency errors which will require additional relaxations. Although higher order interpolations can be used, they are no more effective than the minimal order except for a few special cases [Brandt, 1977]. Thus, minimal order interpolations should generally be used, as they are less complex and just as effective as higher order interpolations. One exception to this rule is when a grid is visited for the very first time. In this case, the interpolation order should be at least $\hat{m} + p$, where p is the order of the approximation scheme [Brandt, 1979]. This higher order interpolation is denoted by I_k^{k+1} . For a second order differential equation discretized with a second order approximation, the appropriate polynomial interpretations (I_k^{k+1} and I_k^{k+1}) are linear and cubic.

For the purpose of describing the interpolation processes, it is assumed that the information being transferred between grids is the estimate of the solution.

The linear interpolation process (I_k^{k+1}) is depicted in Figure 3.1.3 . The process is a two-part procedure where the estimate of the solution (or what-

ever is being transferred) on the coarse grid nodes is first copied onto the corresponding fine grid nodes. Then, for the fine grid nodes without a corresponding coarse grid node, the solution from the two nearest coarse grid nodes is averaged.

The cubic interpolation process (I_k^{k+1}) is shown in Figure 3.1.4 . As before, the interpolation is a two-part process where the estimate of the solution is first copied from the coarse grid nodes to the corresponding fine grid nodes. For those fine grid nodes without a corresponding coarse grid node, a weighted average is obtained from the four nearest coarse grid nodes.

The process of interpolating to a coarser grid (restriction) is denoted by the operator I_{k+1}^k . The restriction process is accomplished by either direct injection or a weighted scheme.

In order to use direct injection, the nodes on G^H must be a subset of G^h, which is usually the case. With direct injection, the values on fine grid nodes corresponding to coarse grid nodes are simply copied onto the coarse grid as depicted in Figure 3.1.5 . Direct injection is well suited for use with problems containing very steep gradients. Even though direct injection is both fast and easily implemented, it does not transfer all of the available information present on the fine grid.

The full weighting scheme, though, does use information present on all fine grid nodes and so, preserves some of the high frequency content of the

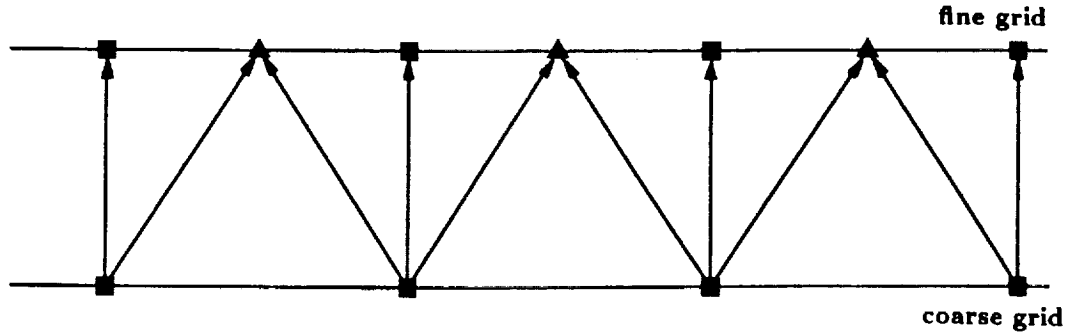

Figure 3.1.3: Linear Interpolation

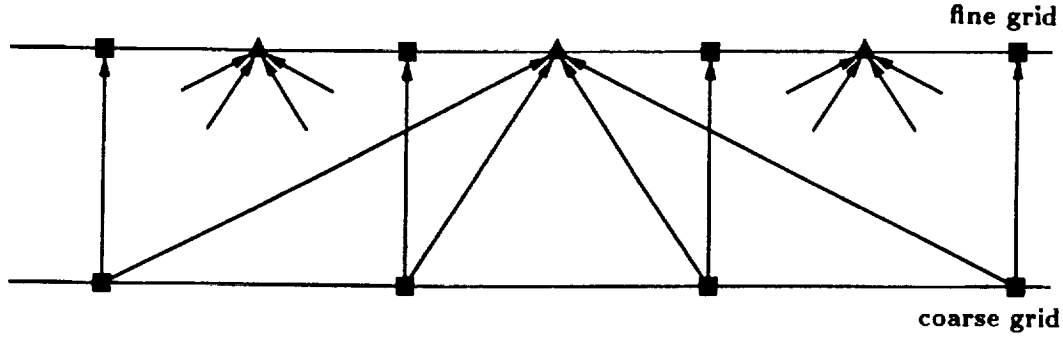

Figure 3.1.4: Cubic Interpolation

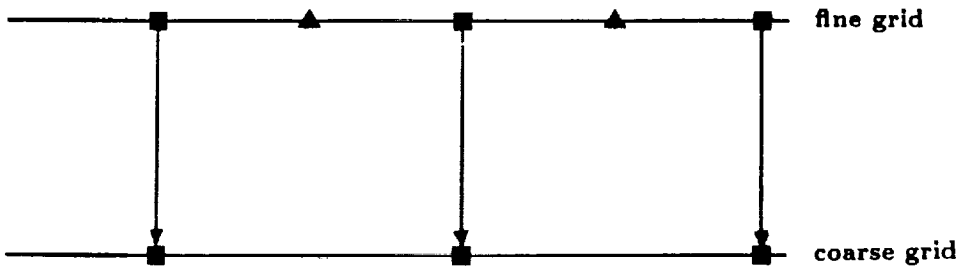

Figure 3.1.5: Direct Injection

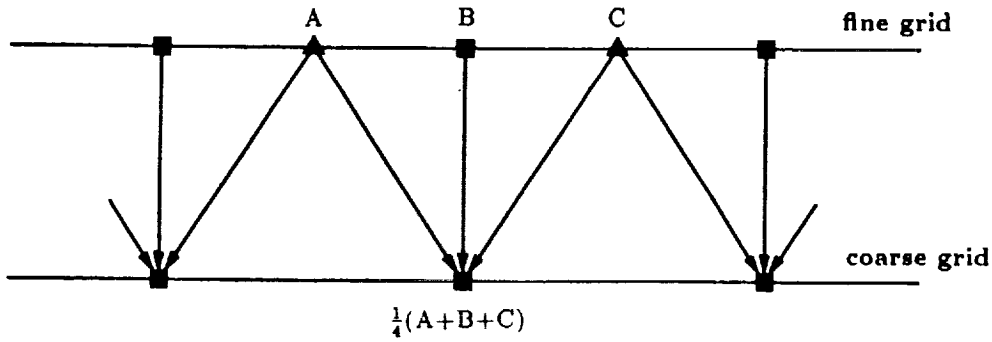

Figure 3.1.6: Weighted Average (full weighting)

finer grid. It does this by employing a weighted average as shown in Figure 3.1.6 to transfer fine grid information onto a coarser grid.

When transferring to a finer level which has already been visited, a "coarse grid correction" step is performed. The correction is designed so as to include the most recent low frequency information in the solution from a coarse grid onto the next finer level,

$$u^k = u^k + I_{k-1}^k(u^{k-1} - I_k^{k-1}u^k). \tag{3.1.13}$$

In transferring to a coarser grid, it is desired to approximate the current fine grid solution on the coarse grid. This is done by calculating a static residual on the fine grid and interpolating it down to the next coarser gird where it is added to the existing coarse grid forcing function to give

$$f^k = L^k u^k - I_{k+1}^k(f^{k+1} - L^{k+1}u^{k+1}). \tag{3.1.14}$$

Such a modification allows for the solution to the coarse grid equations to coincide with the fine grid solution [Brandt, 1979].

A flowchart of a multigrid method, the Full Approximation Scheme (FAS) is shown in Figure 3.1.7 . First, the problem (already discretized onto a sequence of grids) on the coarsest grid is solved by the use of either an iterative or a direct solver. Second, the solution is interpolated to the next finer level using a higher order (cubic) interpolation. Third, residuals (e^k) are computed as a relaxation (Gauss-Seidel) sweep is performed at the current level. The residuals are then used to determine if the solution has converged.

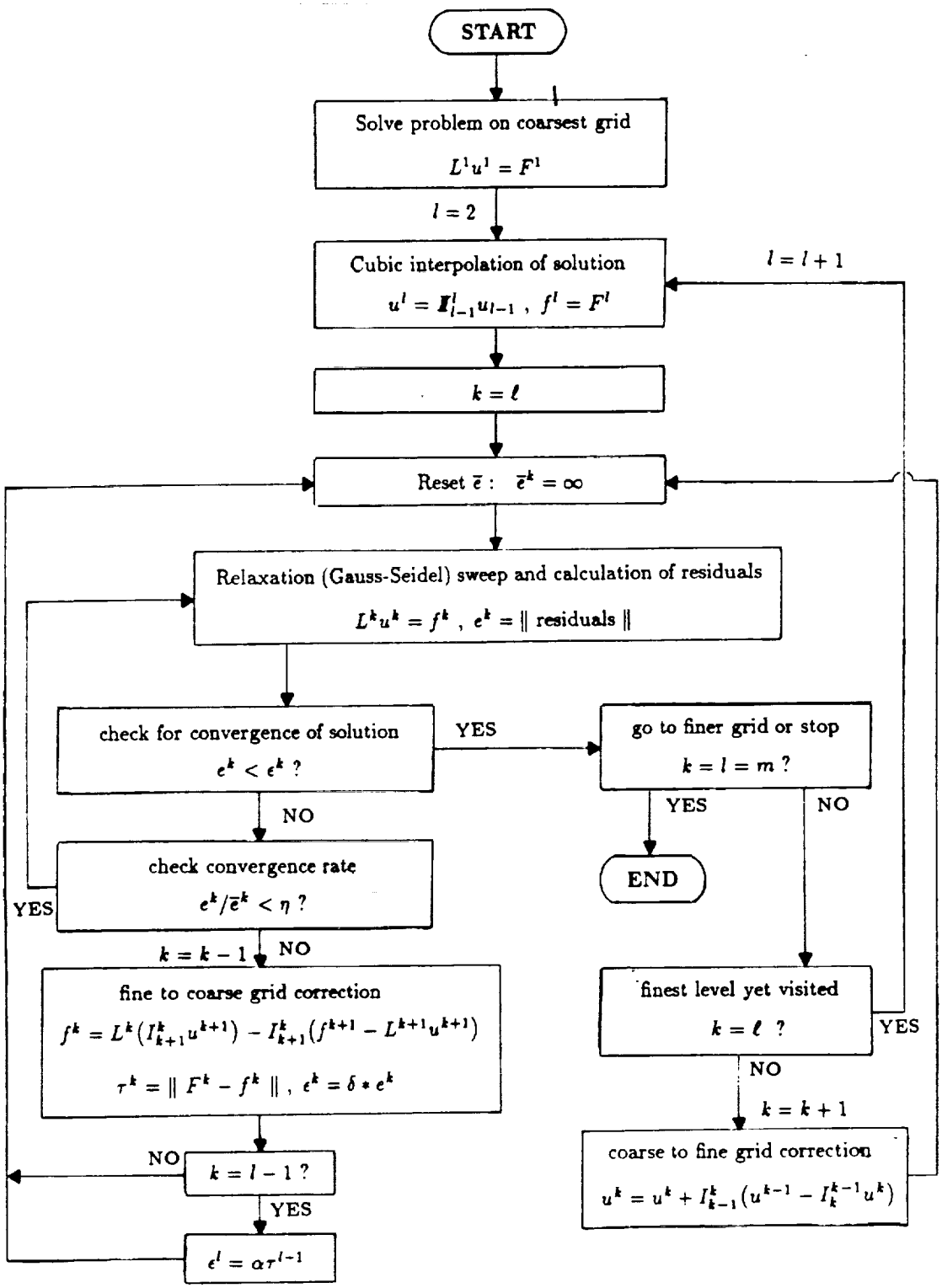

Figure 3.1.7: Multigrid Flowchart

If convergence is detected ($e^k < \epsilon^k$), the multigrid algorithm proceeds to the next finer level or it is terminated if the current level is the finest. If this finer level has been visited before, the coarse grid correction step previously described is performed to correct the fine grid solution.

If the solution has yet to converge, the convergence rate ($e^k/\bar{e}^k$) is checked. If this rate is satisfactory (larger than η), then an additional relaxation sweep at the current level is done by returning to the third step. If the convergence rate is too slow, the solution is interpolated (restricted) down to the next coarser level where a fine to coarse grid correction is performed on the coarse grid forcing function. After the correction is made, execution returns to the third step to begin the relaxation loop for the coarser grid.

The parameters α, δ, and η (included in the multigrid flowchart, Figure 3.1.7) are referred to as "switching parameters" and assist in guiding the flow of the algorithm. These parameters range in value from 0 to 1 . The stopping factor η was discussed earlier in this chapter.

The parameter δ governs the convergence criteria on coarser grids. After interpolating to a coarser grid, the most recent residual error norm (from the finer level) e^{k+1} is reduced by a factor of δ to obtain the convergence criteria ϵ for the current grid level, thus

$$\epsilon^k = \delta e^{k+1}. \tag{3.1.15}$$

The parameter δ is designed such that the errors present on the current grid

are reduced by a factor similar to the reduction achieved on the finer grid per each multigrid cycle. A multigrid cycle consists of the processes involved in performing a few relaxation sweeps on the finest grid, then proceeding down to the coarsest level and returning back to the finest grid level. Since $\overline{\mu}$ is a measure of the convergence rate (per relaxation sweep) of the high frequency errors, it can be used to find δ; hence,

$$\delta = \overline{\mu}^r, \tag{3.1.16}$$

where r is the number of relaxation sweeps on the finer level per multigrid cycle. "With good relaxation schemes $\overline{\mu} \approx 0.5$ and $r \approx 3$," thus setting $\delta = 0.125$ is usually a good idea [Brandt, 1979]. Like η, δ may be found by trial and error, and variations in δ have little effect on multigrid efficiency [Brandt, 1977].

The parameter α is used to determine the convergence criteria ϵ for the finest level visited. "On the currently finest level ($k = l$) we need convergence to within the estimated size of the truncation error" [Brandt, 1977]. If grid level l has already been visited, the current estimate of the truncation error is τ^{l-1}, but an estimate corresponding to level l is desired for use as the convergence criterion. Therefore, the convergence criteria on grid level l may be taken as

$$\epsilon^l = \alpha \tau^{l-1}, \tag{3.1.17}$$

where the parameter α is a scaling factor relating the truncation error from the

coarser grid to that on the finer grid. Since the truncation error is dependant upon the nodal spacing (H, h) and the approximation order (p),

$$\alpha = \left(\frac{H}{h}\right)^{-p}. \tag{3.1.18}$$

Assuming that the nodal spacing on the coarse grid is twice that of the next finer grid (which is usually the case), (3.1.18) may be rewritten as

$$\alpha = 2^{-p}. \tag{3.1.19}$$

Returning to (3.1.17), if τ^{l-2} is not known, τ^{l-2} is used to determine ϵ^l. The new equation is

$$\epsilon^l = \alpha^2 \tau^{l-2} \tag{3.1.20}$$

and is used when grid level l is visited for the first time [Brandt, 1977].

3.2 Adaptive Grid Method

Adaptive grid schemes are similar to multigrid methods. Like multigrid methods, adaptive grid algorithms use a sequence of finer and finer uniform grids to solve a problem. But with adaptive grid schemes, each finer grid may be confined to increasingly smaller subdomains which require additional refinement. The purpose of adaptive grid methods is to minimize the computational work by identifying regions of the domain which have converged to some desired accuracy so that further computations are confined only to those subregions which have yet to converge and, thus, require additional

refinement. Such regions often occur in problems with steep gradients, such as transonic flows, fronts, shocks, etc. [Hedstrom and Rodrique, 1982].

Here, the work is restricted to one-dimensional problems. This allows the Gauss-Seidel relaxation step to be replaced with a line (direct) solver. This eliminates the need to return to the coarser grids since the direct solver solves the problem to the level of truncation error on the associated grid. In order to make full use of the coarser grids, the following adaptive scheme is introduced. With this scheme the finer grids may be confined to increasingly smaller subdomains. The adaptive grid procedure is outlined in Figure 3.2.1, and proceeds by first solving the discretized problem on a coarse grid using a direct solver. Second, the solution is interpolated to the next finer grid level. The problem is now solved (again using a direct solver) on the current level over all existing subdomains or over the entire domain, as the case may be. The relative truncation errors are now found and used to construct the adaptive subgrids so that they contain the portions of the domain which have not converged to the desired accuracy ϵ. Several subgrids may be needed as those portions of the domain which have not converged may lie separated from each other. For such situations, each separate subdomain requiring refinement is allocated to a different subgrid. In order to do this, effective boundary conditions must be specified for each subgrid to ensure the continuation of the solution as each subdomain is treated as a separate problem. The problem is

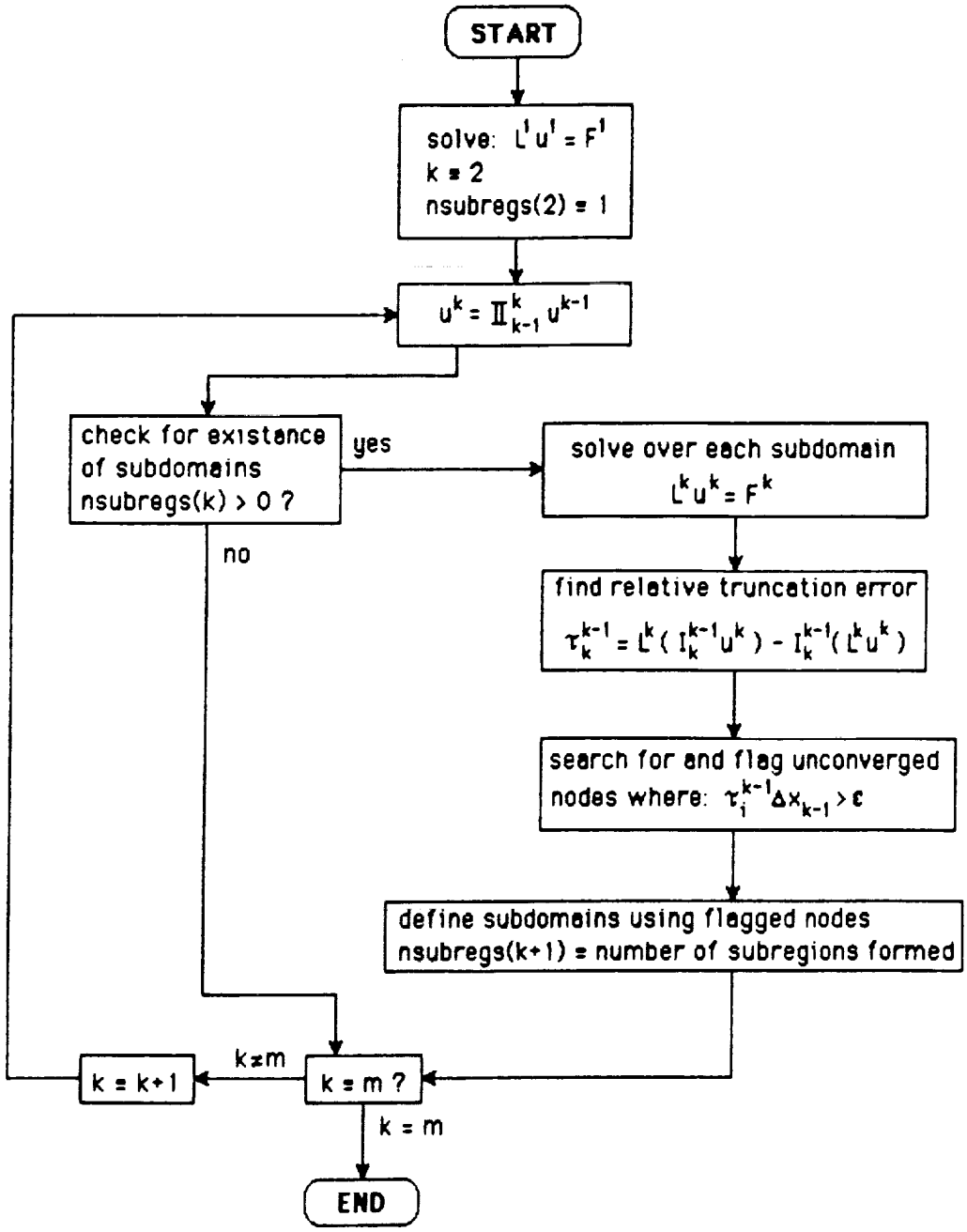

Figure 3.2.1: Adaptive Grid Flowchart

now solved with the direct solver on the next finer grid only over those portions of the subdomain covered by subgrids. At this point, the algorithm returns to the interpolation step and then proceeds to check for the existence of any subgrids on this finer grid level. As the criterion for convergence ϵ is also used to define the subdomains, the existence of any subgrids requiring further refinement indicates that the solution has yet to converge over the entire domain. So, if no new subgrids are defined, the solution has converged to the accuracy sought on all nodes. Convergence is detected when $\Delta x_{k-1}\tau_i^{k-1} < \epsilon$. By multiplying the local truncation error by Δx_{k-1}, the error is weighted with respect to the grid level $k-1$. Weighting the truncation error in this fashion results in an error measurement comparable to that obtained on any other level. With this algorithm, the final solution is presented as it exists on the finest level. If the solution converges over the entire domain on one of the coarser grids, it is then simply interpolated up to the finest grid for output.

A point to note is that since this algorithm is recursive, a subgrid may itself contain several subgrids on finer levels which, in turn, may contain still more subgrids on even finer levels of discretization.

One approach to the construction of the subgrids is to use a measure of the truncation error to identify subregions requiring refinement. The actual truncation error is not known, but it can be approximated by the relative

(local) truncation error τ_h^H which is found by evaluating (3.2.1) .

$$\tau_h^H = L^H(I_h^H u^h) - I_h^H(L^h u^h) \; . \tag{3.2.1}$$

The relative truncation error is the truncation error on a coarse grid (G^H) relative to that on a finer grid (G^h) and approximates the true truncation error on the coarse grid. It may also be viewed as "the error which arises when the fine grid solution is substituted in the coarse grid equation" [Brandt, 1979]. In order to find τ_h^H, two grid levels are required along with the solution on the finer of the two grids. Once τ_h^H is obtained, it is multiplied by the weighting function Δx_H, and then compared to the desired accuracy in order to identify those regions of the domain (nodes) on the coarser grid for which the differential approximation has not converged. So, to obtain the adaptive subgrids on the next finer level $k+1$, the solution from the current level k is used in (3.2.1) to estimate the local truncation error on grid level $k-1$. This estimate of the truncation error is now compared to the desired accuracy ϵ to identify (flag) those nodes (on level $k-1$) requiring further refinement. These flagged nodes are then grouped together to form subgrids. Since these subgrids define subdomains which are then treated as separate problems, appropriate boundary conditions need to be specified for each subgrid. This is done by extending the subgrid (on level $k-1$) to include nodes for which the differential approximation has converged. The resulting subgrids are now defined on the next grid level by identifying the nodes (on level $k+1$) corresponding to the

subgrid boundaries on level $k-2$. To provide an estimate of the higher order derivatives (such as flux) across subdomain boundaries, the subgrids may be extended to include more than just a single converged node (see Figure 3.2.2).

The subgrids are constructed with the idea of avoiding any unnecessary or duplicate computational work so as to increase the efficiency of the algorithm. In constructing adaptive grids for one-dimensional problems, the process of flagging and grouping unconverged nodes is combined with that of defining the subgrid boundaries. This procedure begins by sequentially scanning the nodes in each subgrid on the previous level $(k-1)$ looking for nodes on which the solution has yet to converge to the desired accuracy. The relative truncation error and ϵ are used to determine whether or not the solution has converged on node i. Once an unconverged node is found (using (3.2.2)), it is used to locate the boundary node where the first (or next) subgrid begins:

$$|\Delta x_{k-1} \tau_i^{k-1}| > \epsilon \ . \tag{3.2.2}$$

As the nodes are scanned sequentially, this subgrid boundary node is taken as the previous node (which has converged to the desired accuracy). At this point the corresponding node on grid level $k+1$ is identified as the boundary node at which the subgrid begins. This is accomplished by storing the location of the node (by node number) into an array (IADAPT) containing pointers used to define the subgrid boundaries. The scanning process continues (on level $k-1$) by searching for the next node on which the solution has converged.

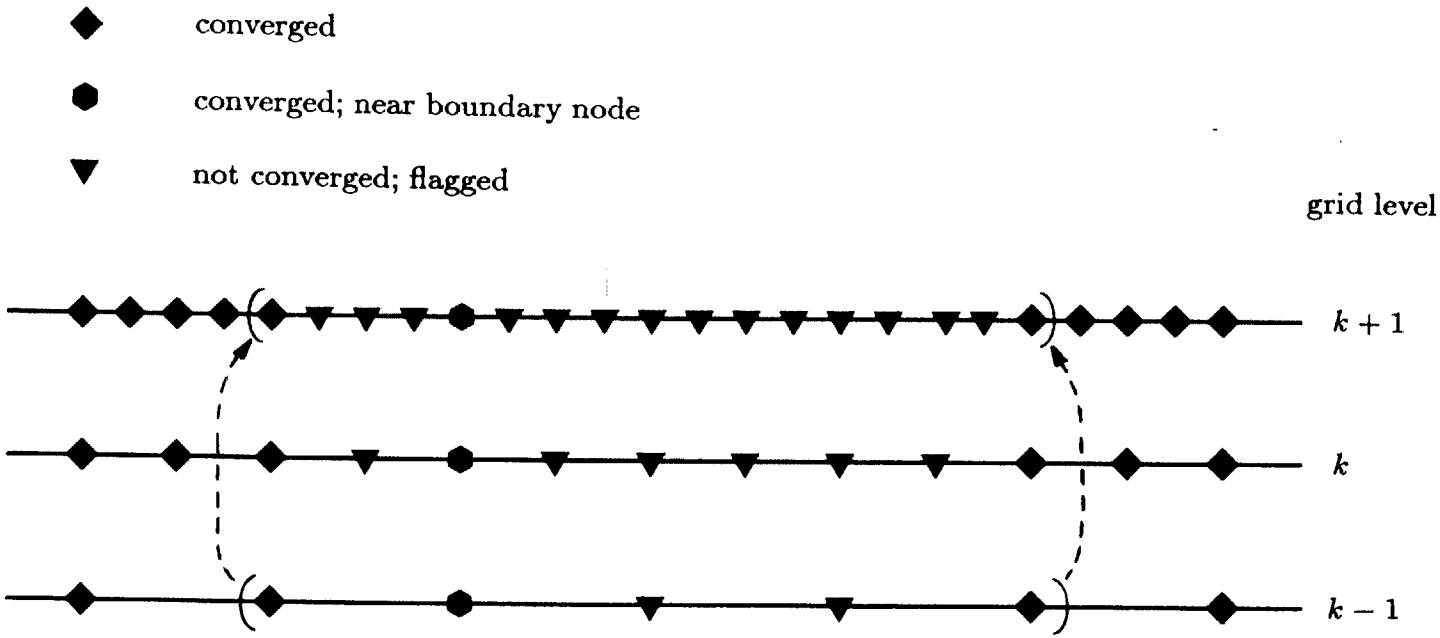

Figure 3.2.2: Adaptive Grids

As before, the relative truncation error and the desired accuracy ϵ are used to make this determination. But since the search is for nodes on which the solution has converged, Equation (3.2.3) is used instead of (3.2.2)

$$|\tau_i^{k-1}| \leq \epsilon. \tag{3.2.3}$$

The first node found on which the solution has converged becomes the boundary at which the subgrid ends. As before, the corresponding subgrid boundary node on level $k+1$ is identified and its location is stored in the array IADAPT. The algorithm now returns to scan the remaining nodes, looking to construct another subgrid where refinement is necessary. The process continues until all the nodes contained within the subgrids on level $k-1$ are searched. The subgrids are now complete and have been identified on the next finer grid level; with the exception of the boundary nodes, the subgrids contain only unconverged nodes. As the solution on the subgrid boundaries has converged to the desired accuracy, the problem within each subgrid may now be solved (using Dirichlet conditions) without further modifications.

3.3 Transient Algorithms

Transient problems are solved using an implicit finite difference discretization scheme in both the multigrid and adaptive grid algorithms presented. The spatial terms of the partial differential equations are discretized onto uniform grids using a central finite difference approximation while the time domain is discretized with a backward finite difference. As usual, the unknown terms are

placed on the left hand side of the equation and the known terms are placed on the right hand side. This leads to an algebraic system of equations of the form

$$L^j U^j = F^{j-1}, \tag{3.3.1}$$

where j and $j-1$ are integer values denoting the time steps of interest. The operator L now includes terms containing the time increment Δt. The term U represents the discrete solution at the current time step and F contains the discretized forcing function (if it exists) along with the solution from the previous time step.

In handling transient problems, local time stepping is not used here; instead, a fixed time increment is employed throughout the solution process. Each time step is treated as an individual problem and is solved separately using the previously described multigrid or adaptive grid methods. In transferring from one time step to the next, the solution from the finest grid (current time step) is coarsened (restricted) and used as the initial estimate for the problem on the coarsest grid at the next time step:

$$u^{j+1}_{coarsest} = I^{coarsest}_{finest} u^j_{finest}, \tag{3.3.2}$$

where j denotes the current time step (just solved) and $j+1$ denotes the next time step. The operator $I^{coarsest}_{finest}$ is simply a fine to coarse restriction operator which is applied the number of times required to restrict the finest

grid solution down to the level of the coarsest grid.

3.4 Nonlinear Adaptive Grid Algorithm

The adaptive grid algorithm is also used to solve nonlinear problems. Discretizing a nonlinear problem results in a set of simultaneous algebraic equations containing coefficients dependant upon the solution. To handle the nonlinearities, Picard iterations are applied to the tridiagonal solver.

The solution to nonlinear problems may contain fronts with very steep gradients. Solving such a problem using the adaptive grid method requires that an estimate of the solution be transferred from a coarse grid to a fine grid. In transferring the solution to a finer grid level, it is desired to minimize the introduction of large errors. Due to the nature of nonlinear problems, such errors can result in numerical instabilities. To limit the creation of these errors, the adaptive grid algorithm uses only linear interpolations. Using a cubic (or higher order) interpolation to transfer the approximation of the solution to a finer grid can result in the introduction of relatively large errors near fronts with steep gradients. These errors are created as the interpolated approximation (on the fine grid) will tend to overshoot the actual solution near fronts. The use of a linear interpolation can avert this potential problem, but it will create some high frequency errors (see Figures 3.4.1 and 3.4.2). The high frequency errors can easily be smoothed out on the fine grid at the expense of some additional computational work.

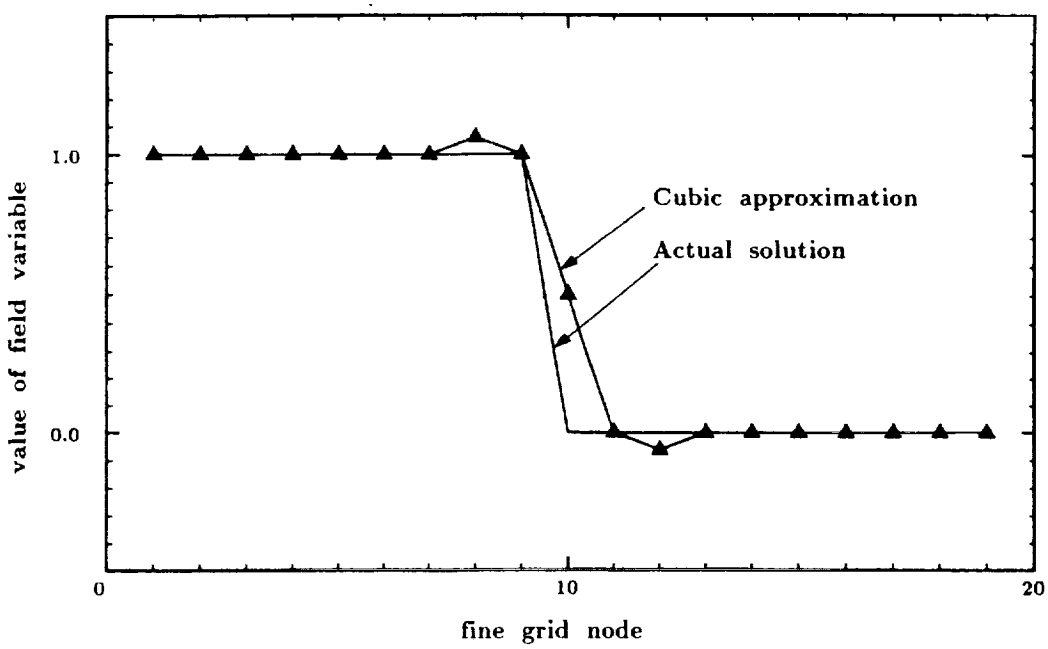

Figure 3.4.1: Cubic Interpolation at a Front

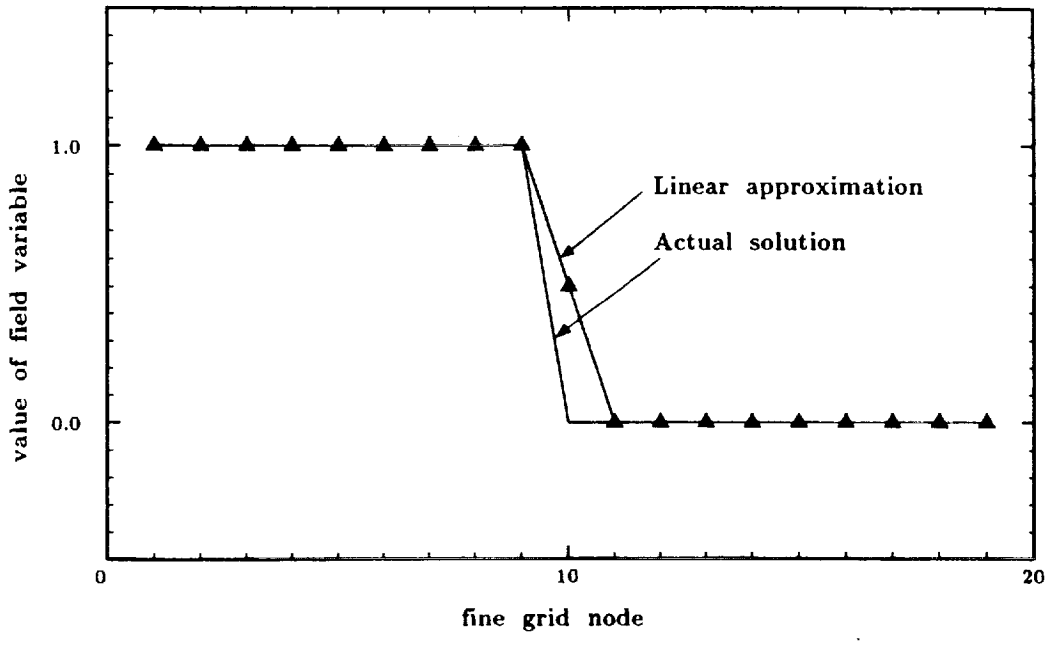

Figure 3.4.2: Linear Interpolation at a Front

To solve nonlinear problems with the adaptive grid method, Picard iterations are used in conjunction with a direct solver. The adaptive grid process may require the use of several subgrids, with each subgrid encompassing a different portion of the domain. Each subdomain is treated as a separate problem. The solution on each subgrid is found by using Picard iterations along with the tridiagonal method. The direct solver is iterated until the approximation converges on the solution. This procedure begins by using the most recent approximation of the solution to estimate the value of the nonlinear coefficients. The coefficients are held constant and the direct linear solver is used to obtain a better estimate of the solution. A residual norm e^k is now computed (over the current subdomain) using

$$e^k = \|F^k - L^k u^k\|. \tag{3.4.1}$$

The nonlinear coefficients are updated and the process is repeated until the residual norm becomes approximately constant from one iteration to the next, signalling that the approximation has converged. Convergence is defined by

$$e^k - \overline{e}^k < toler, \tag{3.4.2}$$

where $\overline{e}^k$ is the residual norm as it existed prior to the relaxation sweep and *toler* is a user defined tolerance used to detect convergence. When (3.4.2) is satisfied, program execution proceeds on to either the next subdomain or the next grid level. The resulting algorithm is presented in Figure 3.4.3 .

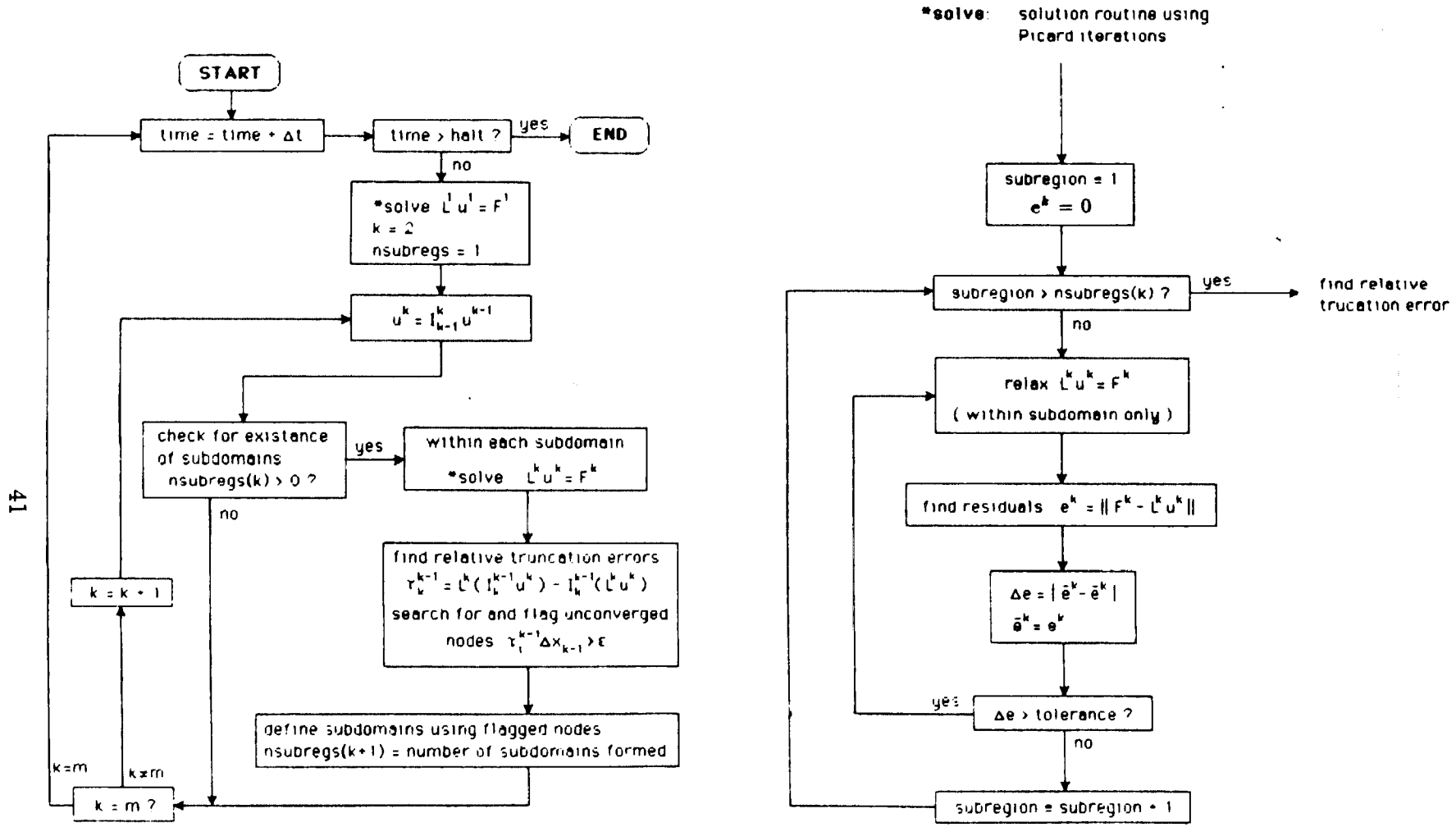

Figure 3.4.3: Nonlinear Adaptive Grid Algorithm

Chapter 4

Test Problems

Several test problems are used to verify the multigrid and adaptive grid algorithms implemented into computer programs. The test problems are also used to highlight some of the main features of each program. Each of the problems are discretized using a central finite difference approximation for the spatial terms. In addition, the time-dependent problems use a backward finite difference approximation for the transient terms. For the approximations used to solve the steady-state problems, the truncation errors are on the order of Δx^2 (also denoted as $O(\Delta x^2)$). The transient test problems are discretized using a finite difference approximation with $O(\Delta x^2, \Delta t)$.

4.1 Test Problem #1

The first test problem was used to verify the initial multigrid program written. This program was first implemented on a personal computer (Apple IIc) and later rewritten for use on a VAX machine. Test problem (4.1.1) is designed to show that the multigrid algorithm quickly solves a problem which may take a very large number of iterations to solve on a fine grid using an iterative relaxation method. The problem is

$$\frac{d^2u}{dx^2} = f(x) \quad (0 \leq x \leq 1), \tag{4.1.1a}$$

subject to the boundary conditions:

$$u(0) = 0 \quad \text{and} \quad u(1) = 0 \ . \tag{4.1.1b}$$

The forcing function $f(x)$ first used with (4.1.1) is

$$f(x) = -\pi^2 \sin(\pi x) \ . \tag{4.1.2}$$

Equation (4.1.1) with its forcing function given by (4.1.2) is referred to as test problem #1a. Discretizing (4.1.1) using (4.1.2) onto a grid containing n nodes yields:

$$\frac{1}{\Delta x^2}(u_{i-1} - 2u_i + u_{i+1}) = -\pi^2 \sin(\pi i \Delta x) \quad (1 < i < n), \tag{4.1.3a}$$

and (for the boundary conditions)

$$u_1 = 0 \quad \text{and} \quad u_n = 0 \ . \tag{4.1.3b}$$

The analytical solution to (4.1.1) with (4.1.2) is

$$u(x) = \sin(\pi x) \ . \tag{4.1.4}$$

A different forcing function (4.1.5) containing both high and low frequency terms is now introduced to illustrate that the effect of low frequency errors is to slow down the convergence rate of the solution:

$$f(x) = -\pi^2 \sin(\pi x) - \frac{(26\pi)^2}{10} \sin(26\pi x) \ . \tag{4.1.5}$$

Equation (4.1.1) with the forcing function defined by (4.1.5) is referred to as test problem #1b. Discretizing (4.1.1) using (4.1.5) results in

$$\frac{1}{\Delta x^2}(u_{i-1} - 2u_i + u_{i+1}) = -\pi^2 \sin(\pi i \Delta x) - \frac{(26\pi)^2}{10} \sin(26\pi i \Delta x) \ , \quad (4.1.6a)$$

where $(1 < i < n)$, and

$$u_1 = 0 \quad \text{and} \quad u_n = 0 \ . \quad (4.1.6b)$$

The analytical solution for this case is

$$u(x) = \sin(\pi x) + 0.1 \sin(26\pi x) \ . \quad (4.1.7)$$

4.2 Test Problem #2

The multigrid program was extended to solve transient one-dimensional partial differential equations (PDEs). A new test problem,

$$\frac{\partial^2 u}{\partial x^2} = \frac{\partial u}{\partial t} \quad (0 \le x \le 1, \ t \ge 0), \quad (4.2.1a)$$

subject to the following conditions:

$$u(0,t) = 0, \ u(1,t) = 0, \quad \text{and} \quad u(x,0) = \sin(\pi x) \quad (4.2.1b)$$

was introduced. In addition to the multigrid program, the adaptive grid method is also used to solve this transient problem. Discretizing the problem using an implicit finite difference scheme for the time domain gives

$$-Mu_{i-1}^j + (2M+1)u_i^j - Mu_{i+1}^j = u_i^{j-1} \quad (1 < i < n, \ j \ge 1), \quad (4.2.2a)$$

where

$$M = \frac{\Delta t}{\Delta x^2} \ , \tag{4.2.2b}$$

with the boundary conditions:

$$u_1^j = 0 \quad \text{and} \quad u_n^j = 0. \tag{4.2.2c}$$

The superscripts j and $j-1$ are integers referring to the current and previous time steps. The initial condition (4.2.1b) is discretized as

$$u_i^0 = \sin(\pi i \Delta x). \tag{4.2.2d}$$

The analytical solution for this problem is

$$u(x,t) = \sin(\pi x) e^{-\pi^2 t} \ . \tag{4.2.3}$$

4.3 Test Problem #3

An upwind/downwind Gauss-Seidel relaxation method was introduced into the multigrid program. The problem solved, Equation (4.3.1), is a one-dimensional convection-diffusion equation with constant coefficients,

$$\kappa \frac{\partial^2 u}{\partial x^2} - \nu \frac{\partial u}{\partial x} = \frac{\partial u}{\partial t} \quad (0 \le x \le 1, \ t \ge 0), \tag{4.3.1a}$$

where $\nu \mathcal{L} >> \kappa$ with the size of the domain $\mathcal{L} = 1$. Equation (4.3.1a) is subject to the conditions

$$u(0,t) = 0, \ u(1,t) = 0, \quad \text{and} \quad u(x,0) = \sin(\pi x). \tag{4.3.1b}$$

The discretized form of this problem is

$$-(M+N)u_{i-1}^{j}+(2M+1)u_{i}^{j}-(M-N)u_{i+1}^{j}=u_{i}^{j-1}\quad(1<i<n,\ j\geq 1), \tag{4.3.2}$$

where

$$M=\frac{\kappa\Delta t}{\Delta x^2},\quad\text{and}\quad N=\frac{\nu\Delta t}{2\Delta x}. \tag{4.3.3}$$

As in the previous problem, the boundary and initial conditions are discretized as

$$u_1^j=0,\quad u_n^j=0,\quad\text{and}\quad u_i^0=\sin(\pi i\Delta x). \tag{4.3.4}$$

The analytical solution to this problem is

$$u(x,t)=4a\pi^2e^{ax}\sum_{m=1}^{\infty}\frac{m}{\beta_m\gamma_m}\left[(-1)^me^{-a}-1\right]e^{-\lambda_m t}\sin(m\pi x), \tag{4.3.5}$$

where

$$\beta_m=a^2+(m-1)^2\pi^2, \tag{4.3.6}$$

$$\gamma_m=a^2+(m+1)^2\pi^2, \tag{4.3.7}$$

$$\lambda_m=\frac{(2mk\pi)^2+\nu^2}{4\kappa}, \tag{4.3.8}$$

$$a=\frac{\nu}{2\kappa}\ . \tag{4.3.9}$$

Test problem #3 is one that can possess rather steep gradients in the field variable and may require a fine mesh to properly resolve the steep gradients present. In addition, an adaptive grid program employing a direct solver was also used to solve the problem.

4.4 Test Problem #4

The next test problem is the nonlinear viscous Burgers' equation (4.1.1) which is solved using adaptive grids,

$$\nu \frac{\partial^2 u}{\partial x^2} - u \frac{\partial u}{\partial x} = \frac{\partial u}{\partial t} \quad (0 \leq x \leq 1,\ t \geq 0), \tag{4.4.1}$$

subject to the conditions:

$$u(0,t) = 1,\ u(1,t) = 0, \quad \text{and} \quad u(x,0) = 0. \tag{4.4.2}$$

Discretizing (4.4.1) yields:

$$-(M + N_i^j)u_{i-1}^j + (2M+1)u_i^j - (M - N_i^j)u_{i+1}^j = u_i^{j-1} \quad (1 < i < n,\ j \geq 1), \tag{4.4.3a}$$

where

$$M = \frac{\nu \Delta t}{\Delta x^2}, \qquad N_i^j = \frac{\Delta t}{2\Delta x} u_i^j\ . \tag{4.4.3a, b}$$

The nonlinearity causes a problem because the coefficient N_i^j in (4.4.3a) is not known since it depends upon the solution sought (u_i^j). To get around this problem, the most recent approximation of the solution $\overline{u}_i^j$ available (from the previous grid level or iteration) is used instead of u_i^j in (4.4.4).

$$N_i^j = \frac{\Delta t}{2\Delta x} \overline{u}_i^j \tag{4.4.4}$$

Having approximated N_i^j, the discretized set of equations may now be solved. The new solution can then be used to find a better approximation for the

coefficient N_i^j, which in turn can be used to compute a more accurate estimate of the solution and so on – until the solution ceases to change by some predefined amount. Thus, this discretization scheme leads to an iterative solution method. Usually, only a few iterations are required for the solution to converge for each time step.

Summarizing, we have

$$-(M + N_i^j)u_{i-1}^j + (2M + 1)u_i^j - (M - N_i^j)u_{i+1}^j = u_i^{j-1} \quad (1 < i < n,\ j \geq 1), \tag{4.4.5}$$

where

$$M = \frac{\nu \Delta t}{\Delta x^2}, \qquad N_i^j = \frac{\Delta t}{2\Delta x}\overline{u}_i^j\ , \tag{4.4.5a, b}$$

with

$$u_0^j = 1, \quad u_n^j = 0, \quad \text{and} \quad u_i^0 = 0\ . \tag{4.4.6}$$

For comparative purposes, this problem is also solved using a tridiagonal solver which incorporates Picard iterations. To obtain an equitable comparison between the adaptive grid and tridiagonal methods, both algorithms are used to obtain solutions of similar accuracy. This is done by first solving the problem using the adaptive grid algorithm, and then computing a residual norm e_{AG} (on the finest grid level) using

$$e_{AG} = \|F - Lu\|, \tag{4.4.7}$$

where the subscript AG refers to the adaptive grid scheme. As the tridiagonal

scheme uses the same type of residual to detect convergence of the solution, e_{AG} is used as the convergence criteria for the tridiagonal algorithm.

4.5 Test Problem #5

The fifth test problem is to solve a nonlinear equation modeling one-dimensional water flow in an unsaturated soil. The equation used is the water content formulation of the Richards' equation:

$$\frac{\partial \theta}{\partial t} + \frac{\partial}{\partial x}(K - \frac{K}{C}\frac{\partial \theta}{\partial x}) = 0, \tag{4.5.1}$$

where $C \equiv -\frac{\partial \theta}{\partial h}$, θ is the volumetric water content, K is the hydraulic conductivity, h is tension, t is time, and x is depth measured from the soil surface. The coefficient C is found from the water retention curve

$$S_e = \frac{\theta - \theta_r}{\theta_s - \theta_r} = \left(1 + (\tilde{\alpha}h)^{\tilde{n}}\right)^{-\tilde{m}} , \tag{4.5.2}$$

where

$$\tilde{m} = 1 - \frac{1}{\tilde{n}} . \tag{4.5.3}$$

The hydraulic conductivity is given by

$$K = K_s S_e^{\frac{1}{2}} \left(1 - (1 - S_e^{\frac{1}{\tilde{m}}})^{\tilde{m}}\right)^2 . \tag{4.5.4}$$

In the above equations, K_s is the saturated hydraulic conductivity, θ_r and θ_s are the residual and saturated water contents, and $\tilde{\alpha}$, $\tilde{m}$, and $\tilde{n}$ are model parameters determined from laboratory data. By differentiating and manipulating (4.5.4), a function for $\frac{\partial \theta}{\partial h}$ in terms of θ is found

$$\frac{\partial \theta}{\partial h} = -\tilde{\alpha}\tilde{n}\tilde{m}(\theta_s - \theta_r)\left(S_e^{-\frac{1}{\tilde{m}}} - 1\right)^{\tilde{m}} S_e^{\frac{\tilde{m}+1}{\tilde{m}}} . \tag{4.5.5}$$

And so, C is now given by

$$C = \tilde{\alpha}\tilde{n}\tilde{m}(\theta_s - \theta_r)\left(S_e^{-\frac{1}{\tilde{m}}} - 1\right)^{\tilde{m}} S_e^{\frac{\tilde{m}+1}{\tilde{m}}} \ . \tag{4.5.6}$$

Equation (4.5.1) is solved over a domain of $0 \leq x \leq 700$ cm. and is subject to the boundary conditions:

$$q(0,t) = 1.82 \quad \text{cm/hr} \ \ \text{(flux), and} \quad \theta(700,t) = \theta_{initial} \ . \tag{4.5.7a,b}$$

The initial condition is given in terms of capillary tension and is

$$h_{initial} = -50,000 \quad \text{cm } H_2O \ . \tag{4.5.8}$$

The initial water content $\theta_{initial}$ is computed by introducing the initial tension into the water retention curve (4.5.2). The entire problem is solved up to a times of 5, 15, and 35 days. At 35 days, the infiltration front has not reached $x = 700$ cm, thus (4.5.7b) is still valid.

The soil used for this numerical model is loamy sand, which is characterized by the following parameters:

$$\theta_r = 0.0828, \quad \theta_s = 0.3209, \quad K_s = 270.1 \quad \text{cm/day} \ ,$$

$$\tilde{\alpha} = 0.05501, \quad \text{and} \quad \tilde{n} = 1.5093 \ .$$

The value of K_s was determined from laboratory data while the remaining parameters are taken as the average values for the loamy sand found at the Las Cruces trench site [Hills et al., 1989a,b].

This problem is solved using an adaptive grid method similar to that used to solve the previous problem. The expanse of each of the finer adaptive grids is determined by comparing the estimate of the local truncation errors to a predefined accuracy term ϵ. For comparative purposes, the problem is also solved using the tridiagonal method. In order to obtain a solution of similar accuracy, the value of the residual norm from the adaptive grid method e_{AG} is used as the convergence criteria for the tridiagonal scheme. In both cases, Picard iterations are employed in order to handle the nonlinearities. Each Picard iteration consists of first, estimating the value of the coefficient C and the hydraulic conductivity K using the most recent estimate of θ. The updated values of C and K are then used to compute a better estimate of θ. This process is repeated until convergence is reached, for each time step.

Discretizing (4.5.1) yields

$$
\begin{aligned}
&- MD_{i-\frac{1}{2}}\theta^j_{i-1} + \big(M(D_{i-\frac{1}{2}} + D_{i+\frac{1}{2}}) + 1\big)\theta^j_i \\
&- MD_{i+\frac{1}{2}}\theta^j_{i+1} + \frac{\Delta t}{\Delta x}(K_i - K_{i-1}) = \theta^{j-1}_i \ , \qquad (4.5.9)
\end{aligned}
$$

where

$$
D_{i-\frac{1}{2}} = \frac{1}{2}(D_{i-1} + D_i) \ , \qquad D_{i+\frac{1}{2}} = \frac{1}{2}(D_i + D_{i+1}) \ ,
$$

$$
D_i = \frac{K_i}{C_i} \ , \qquad M = \frac{\Delta t}{\Delta x^2} \ ,
$$

$$
K_i = K_s S_{e_i}^{\frac{1}{2}}\big(1 - (1 - S_{e_i}^{\frac{1}{\tilde{m}}})^{\tilde{m}}\big)^2 \ ,
$$

$$
C_i = \tilde{\alpha}\tilde{n}\tilde{m}(\theta_s - \theta_r)\big(S_{e_i}^{-\frac{1}{\tilde{m}}} - 1\big)^{\tilde{m}} S_{e_i}^{\frac{\tilde{m}+1}{\tilde{m}}} \ ,
$$

and

$$S_{e\,i} = \frac{\overline{\theta}_i - \theta_r}{\theta_s - \theta_r} .$$

The term $\overline{\theta}_i$ is the most recent approximation of the solution on node i for the current time step. For this problem the finite difference nodes are centered in the grid volumes as shown in Figure 4.5.1 . To center the nodes in the grid volumes, the finite difference grids are shifted $\frac{\Delta x}{2}$ into the domain. This is performed on all grid levels so that coarse grid nodes correspond to their fine grid counterparts. Therefore, the direct injection and interpolation processes are performed in the same manner as denoted in chapter four. Applying the flux boundary condition (4.5.7a) to the finite difference equation for node 1 gives

$$(MD_{1\frac{1}{2}} + 1)\theta_1^j - MD_{1\frac{1}{2}}\theta_2^j = \theta_1^{j-1} - \frac{\Delta t}{\Delta x}(K_{1\frac{1}{2}} - q(0,t)) . \qquad (4.5.10)$$

Discretizing the other boundary condition (4.5.7b) results in:

$$\theta_n = \theta_{initial} \qquad (4.5.11)$$

where the subscript n refers to the final node in the grid.

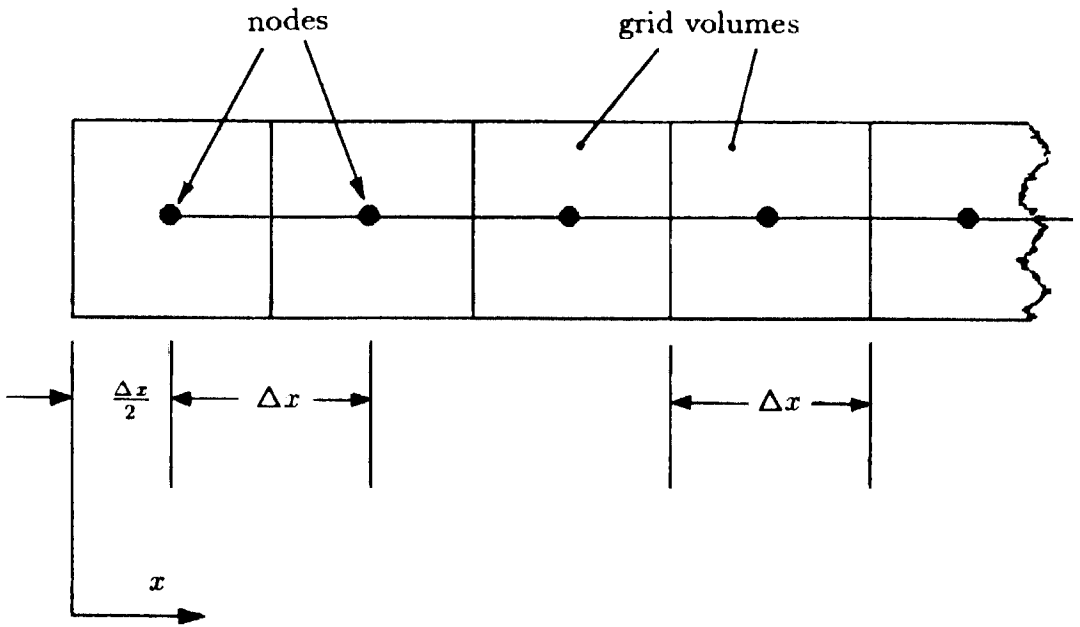

Figure 4.5.1: Finite Difference Discretization with Nodes Centered in Grid Volumes

Chapter 5

Results

The test problems presented in the previous chapter are solved using either the multigrid and/or the adaptive grid methods. The computed solutions are compared to the analytical solution to verify the programs. For comparative purposes, the performance of these algorithms are compared to that of a direct solver (employing the tridiagonal method) and a program using the Gauss-Seidel method.

The performance of the algorithms is measured by comparing the computational work or CPU time (actual running time of the program) required to solve a problem. For the Gauss-Seidel method, the computational work is directly related to the number of iterations needed to solve the problem. For the multigrid and adaptive grid algorithms, the computational work is measured in terms of work units (WU), where one WU is equivalent to a single Gauss-Seidel sweep on the finest grid level. This definition of work units neglects the overhead associated with mapping to different grid levels in both the multigrid and adaptive grid programs. An alternate measure of a programs performance is the CPU time required to solve a problem. Measuring the time needed to solve a problem accounts for all the computations done. A potential complication is that CPU time is machine dependant. Thus, in

order to compare algorithms by using CPU time, the programs must be run on the same computer.

5.1 Results for Test Problem #1

Test problem #1 (4.1.1) is solved using the multigrid method, the Gauss-Seidel algorithm, and a direct solver (tridiagonal method). To compare the performance of these methods, problem #1a,

$$\frac{d^2u}{dx^2} = -\pi^2 \sin(\pi x) \ ,$$

and problem #1b,

$$\frac{d^2u}{dx^2} = -\pi^2 \sin(\pi x) - \frac{(26\pi)^2}{10} \sin(26\pi x) \ ,$$

are discretized and solved (with the tridiagonal and Gauss-Seidel methods) on a grid containing 129 nodes. For the multigrid method, these test problems are discretized using 6 grid levels such that the finest grid contains 129 nodes. With this discretization scheme, there are only 5 nodes on the coarsest grid. For these problems, the multigrid method is many times faster than Gauss-Seidel yet somewhat slower than the direct solver (see Tables 5.1.1 and 5.1.2).

The multigrid solution for test problem #1a, where $f(x)$ given by (4.1.2), is plotted along with its analytical solution in Figure 5.1.1 . In solving this problem, the switching parameters used by the multigrid program are: $\alpha = 0.25$, $\delta = 0.22$, and $\eta = 0.625$. The multigrid program solves problem #1a in 16.35 work units. In contrast, the Gauss-Seidel algorithm requires 22542

Table 5.1.1: Results for Test Problem #1a

Solution Method	Computational Work	CPU Time
Multigrid	16.35 WU	0.08 sec
Gauss-Seidel	22542 iterations	48.41 sec
Direct Solver	–	0.02 sec

Table 5.1.2: Results for Test Problem #1b

Solution Method	Computational Work	CPU Time
Multigrid	18.60 WU	0.07 sec
Gauss-Seidel	22541 iterations	49.69 sec
Direct Solver	–	0.02 sec

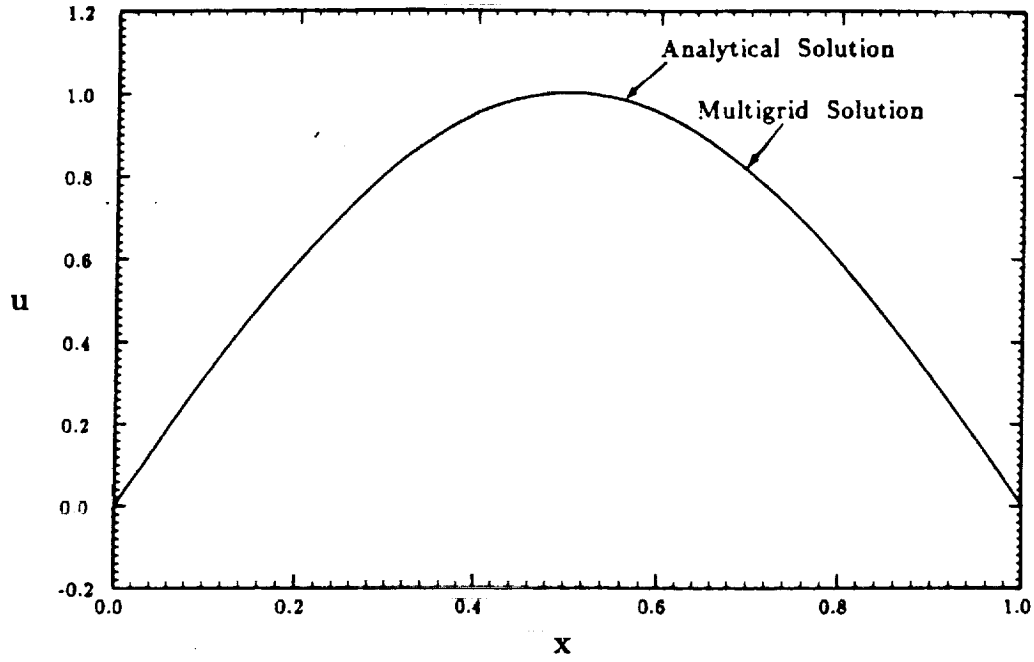

Figure 5.1.1: Solution to Problem #1a

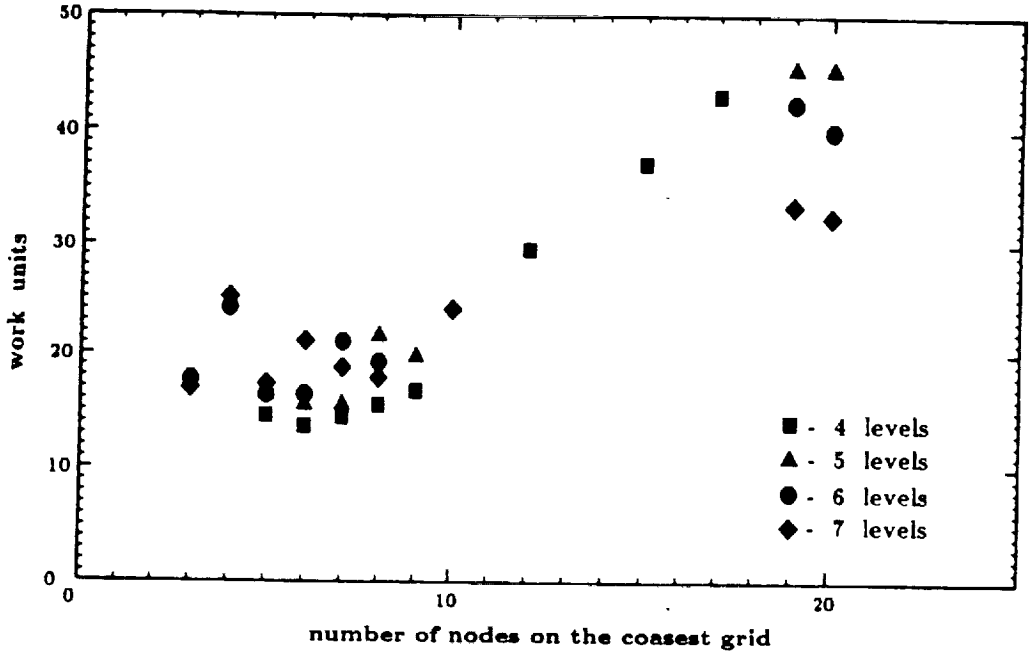

Figure 5.1.2: Effects of Varying the Number of Multigrid Levels

iterations to get the same solution. So, based on work units the multigrid method solves this problem about 1300 times faster than the Gauss-Seidel algorithm. In making this comparison, one should keep in mind that the definition of 'work units' used neglects the overhead involved with the multigrid scheme. A better comparison may be obtained by considering the amount of CPU time required to solve the problem. The Gauss-Seidel algorithm took 48.41 seconds of CPU time (CPU seconds) to solve the problem while the multigrid method needed only 0.08 CPU seconds. That's an approximate increase in speed of 600 times over the Gauss-Seidel scheme. Test problem #1a was solved with a direct solver (tridiagonal method) in 0.02 CPU seconds or about 4 times faster than the multigrid method.

The multigrid program was also used to solve problem #1a several times while varying the number of grid levels and the number of nodes present on the finest grids. The resulting data (CPU time and work units needed to get the solution) is given in Table 5.1.3 . By keeping the number of grid levels constant and varying the number of nodes on the finest level (by increasing the number of nodes on the coarsest level) the effects of the discretization on the coarsest grid is seen in terms of program efficiency. As the number of nodes on the coarsest grid increases, the number of work units required to solve the problem also increases (see Figure 5.1.2). The main reason for this effect is that as a finer and finer grid is used on the coarsest grid level, an increasing

Table 5.1.3: Test Problem #1a, Results

# of levels	# of nodes coarsest grid	# of nodes finest grid	Work Units	CPU time seconds
4	8	57	15.47	0.04
5	8	113	21.64	0.08
6	8	225	19.25	0.13
7	8	449	17.83	0.24
8	8	897	17.04	0.43
4	7	49	14.41	0.04
5	7	97	15.48	0.06
6	7	193	21.06	0.13
7	7	385	18.73	0.21
7	7	769	17.45	0.37
4	6	41	13.59	0.02
5	6	81	15.45	0.06
6	6	161	15.89	0.08
7	6	321	21.12	0.20
8	6	641	18.64	0.35
9	6	1281	22.37	0.80
4	5	33	14.53	0.03
5	5	65	16.36	0.05
6	5	129	16.35	0.08
7	5	257	17.27	0.13
8	5	513	17.14	0.26
9	5	1025	17.61	0.50
10	5	2049	21.83	1.27
6	4	97	24.08	0.09
7	4	193	25.01	0.16
8	4	385	26.52	0.30
9	4	769	26.79	0.59
10	4	1539	26.41	1.16

Table 5.1.3: Test Problem #1a, Results, continued

# of levels	# of nodes coarsest grid	# of nodes finest grid	Work Units	CPU time seconds
6	3	65	17.66	0.05
7	3	129	16.86	0.08
8	3	257	17.42	0.13
9	3	513	17.16	0.25
10	3	1025	17.58	0.50
11	3	2049	21.79	1.29
2	65	129	352.80	0.67
3	33	129	127.80	0.26
4	17	129	43.00	0.12
5	9	129	19.76	0.08
6	5	129	16.35	0.08
7	3	129	16.86	0.08
6	65	2049	237.40	6.24
7	33	2049	56.93	2.09
8	17	2049	28.39	1.50
9	9	2049	16.64	0.96
10	5	2049	21.83	1.27
11	3	2049	21.78	1.27

amount of computational work is required to resolve the lowest frequency errors no longer visible on the coarsest grid. Thus, more computational work is required to resolve these low frequency errors.

The multigrid solution for test problem #1b (in which $f(x)$ is given by (4.1.5)) is presented along with its analytical solution (4.1.7) in Figure 5.1.3 . As with the previous problem, the multigrid method was used to solve problem #1b using the same switching parameters ($\alpha = 0.25, \delta = 0.22$, and $\eta = 0.625$). With this set of switching parameters, an insufficient amount of work is performed on the coarser grids to adequately smooth out the low frequency errors (see Figure 5.1.4). With this problem, the high frequency terms of the solution are aliased and appear as low frequency terms on the coarser grids. So, as the problem is relaxed on the coarse grids, the aliased terms are smoothed out and consequently a low frequency error is introduced into the solution.

One way the errors appearing in Figure 5.1.4 may be reduced is by discretizing the problem (at all levels) onto much finer grids. However, doing so tends to be contrary to the purpose of the multigrid method because an acceptable solution is obtained at the expense of computational work. Therefore, this is not such a good idea due to the large increase in computational work needed to get a good solution.

By exploiting the switching parameters δ and η which are built into the

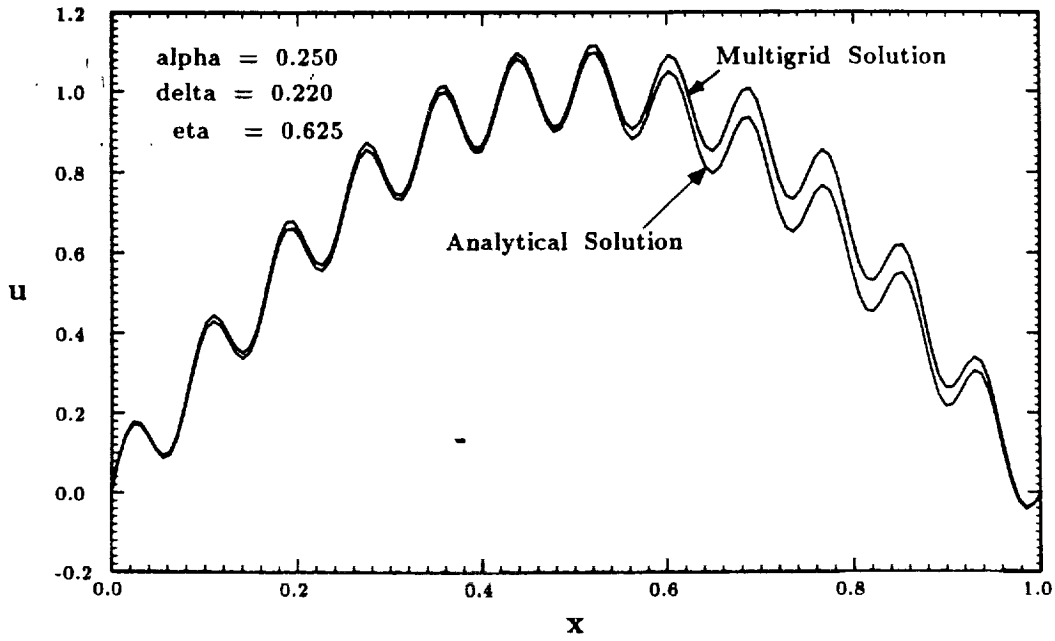

Figure 5.1.3: Solution to Problem #1b, Using Original Switching Parameters

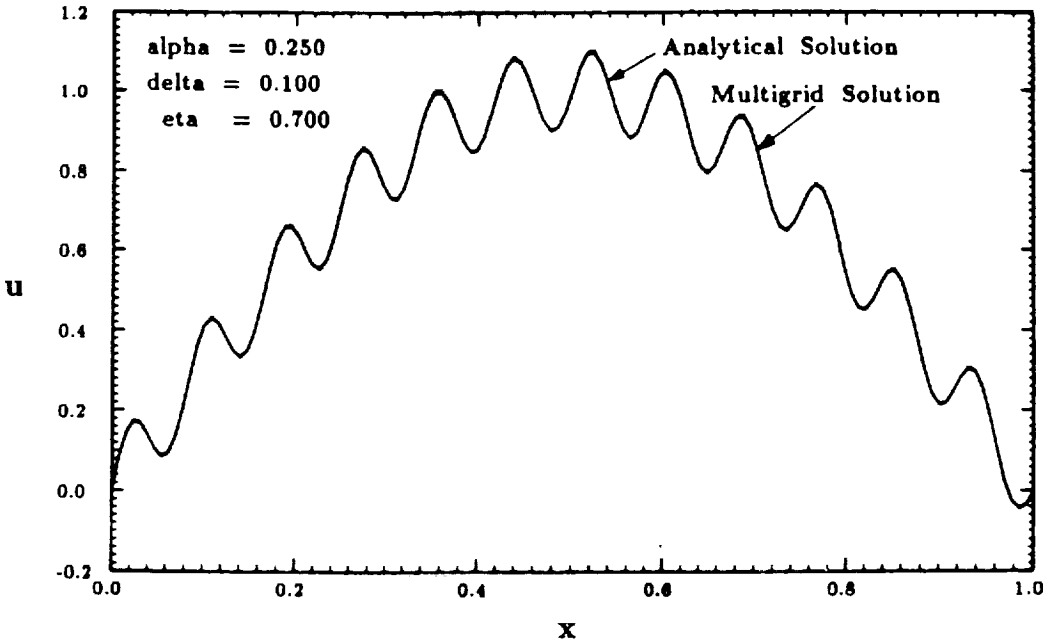

Figure 5.1.4: Solution to Problem #1b, Using Alternate Switching Parameters

multigrid scheme, the problem is easily remedied. Reducing the value of δ results in a more stringent convergence criterion on the coarser grids and thus, more work is performed on these grids. This serves to better resolve the low frequency errors present in the solution on the coarse grids, and so, these errors tend to be less of a problem. Increasing η leads to more relaxation sweeps (per multigrid cycle) on each level before transferring program execution to the next coarser grid. Since more relaxations are performed before changing grid levels, aliasing errors, as well as any errors introduced by the interpolation routines, benefit from the additional relaxation sweeps. Setting $\delta = 0.1$ and $\eta = 0.7$ (values found by trial and error) eliminates the problem of the undesired low frequency error (see Figure 5.1.5) while the number of work units required to get the solution increases slightly. By using this new set of switching parameters, the more accurate solution shown in Figure 5.1.3 is obtained.

The performance of the multigrid algorithm is similar to that obtained for problem #1a. Test problem #1b is solved with the multigrid algorithm in 18.6 WU and takes 0.07 CPU seconds. The Gauss-Seidel method takes 22541 iterations and 49.41 CPU seconds to get the same answer while the the direct solver requires only 0.02 seconds of CPU time. Thus, the multigrid method is about 1200 times faster than Gauss-Seidel but 3.5 times slower than the tridiagonal method.

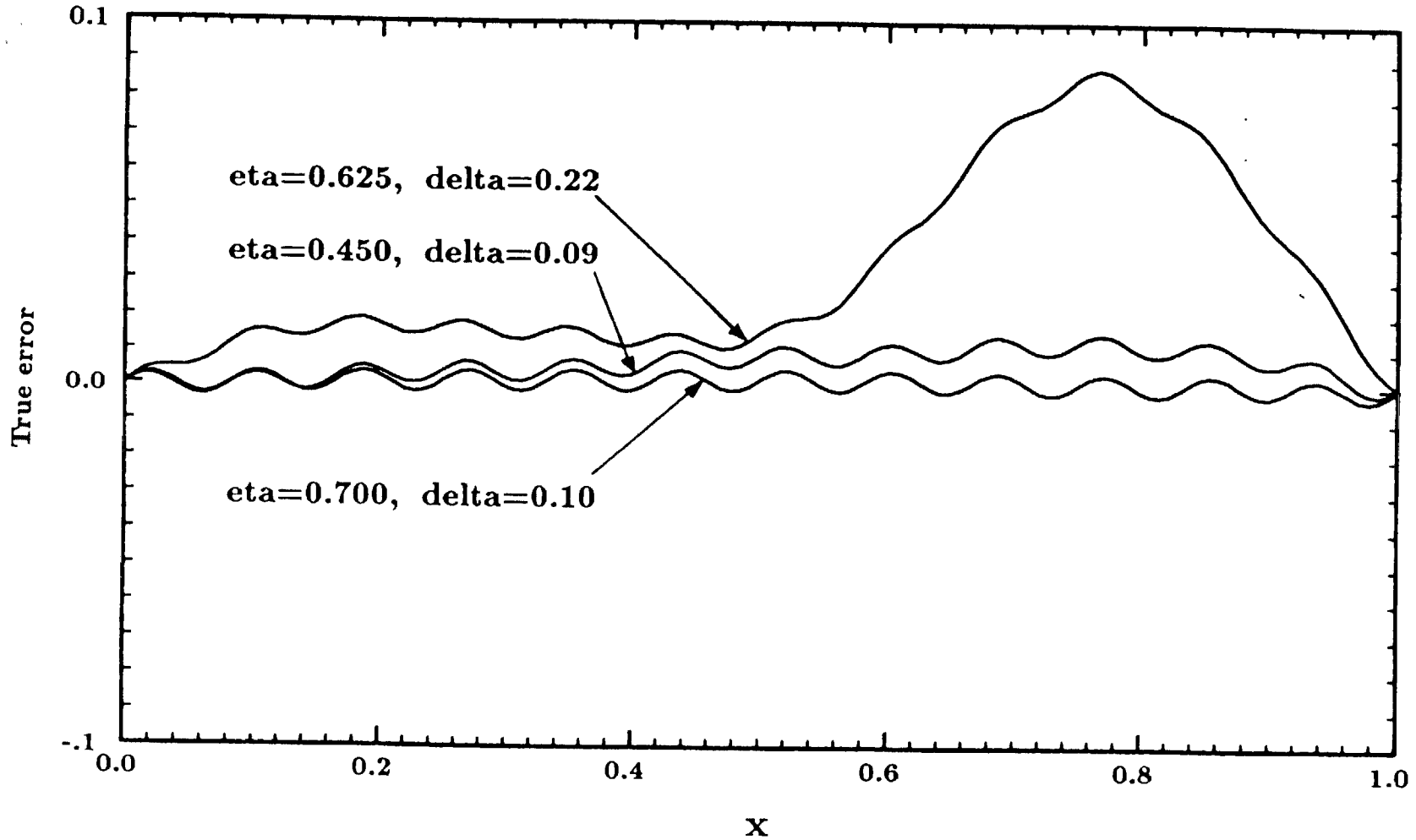

Figure 5.1.5: Effects of Varying the Switching Parameters

5.2 Results for Test Problem #2

The next problem solved, test problem #2, is the one-dimensional transient diffusion equation given by (4.2.1). The problem is discretized onto a grid with 129 nodes and is solved using the multigrid, Gauss-Seidel, tridiagonal and adaptive grid methods. The problem is solved to within 99% of steady state, which occurs at about $t = 0.50$. As with all the transient problems solved here, time stepping with a fixed increment is employed throughout the solution process for each algorithm. The time increment used in solving this problem is $\Delta t = 0.001$. To solve to near steady state ($t = 0.50$), 500 time steps are needed. In order to solve test problem #2 using the multigrid and adaptive grid methods, 6 grid levels with the coarsest grid containing 5 nodes and the finest grid containing 129 nodes are used.

The multigrid and analytical solutions for various x positions ($x = 0.125,\ 0.25,\ 0.50$) are plotted in Figure 5.2.1 . Likewise, in Figure 5.2.2, the adaptive grid solution is presented along with the analytical solution for the same x positions. The switching parameters used in the multigrid program to solve the problem are $\alpha = 0.25, \delta = 0.125$, and $\eta = 0.50$. The multigrid method requires a total of 2212 WU (an average of 4.42 work units per time step) to solve the problem in 7.82 CPU seconds. To investigate the multigrid algorithm, several values of δ and η were used to solve problem #2. The results obtained are tabulated in Table 5.2.1 in terms of computational work

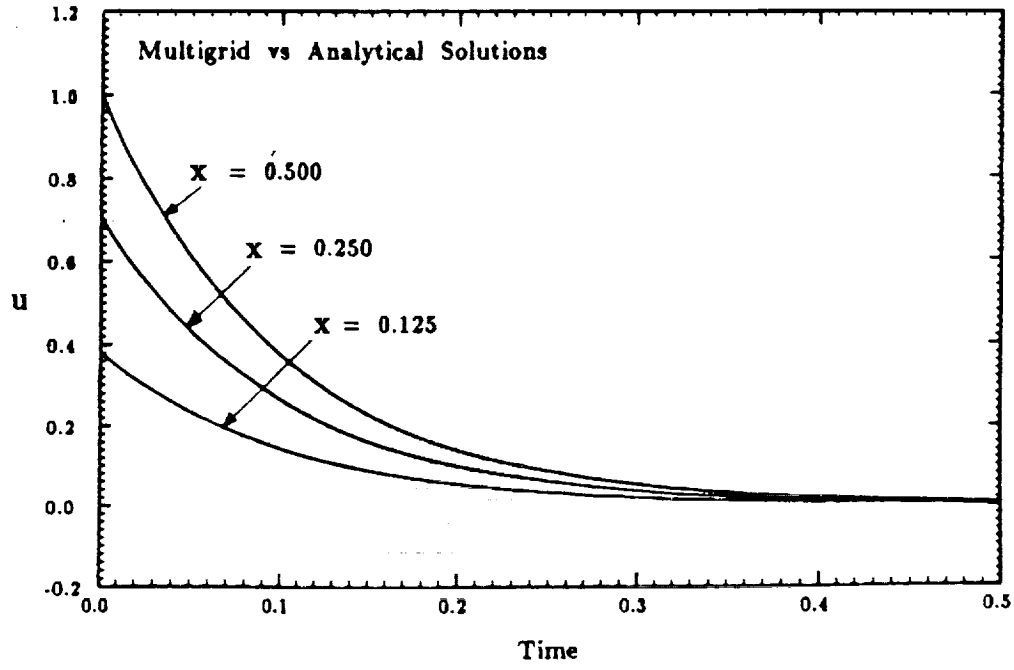

Figure 5.2.1: Multigrid Solution to Problem #2; Analytical and Multigrid Solutions Overlie

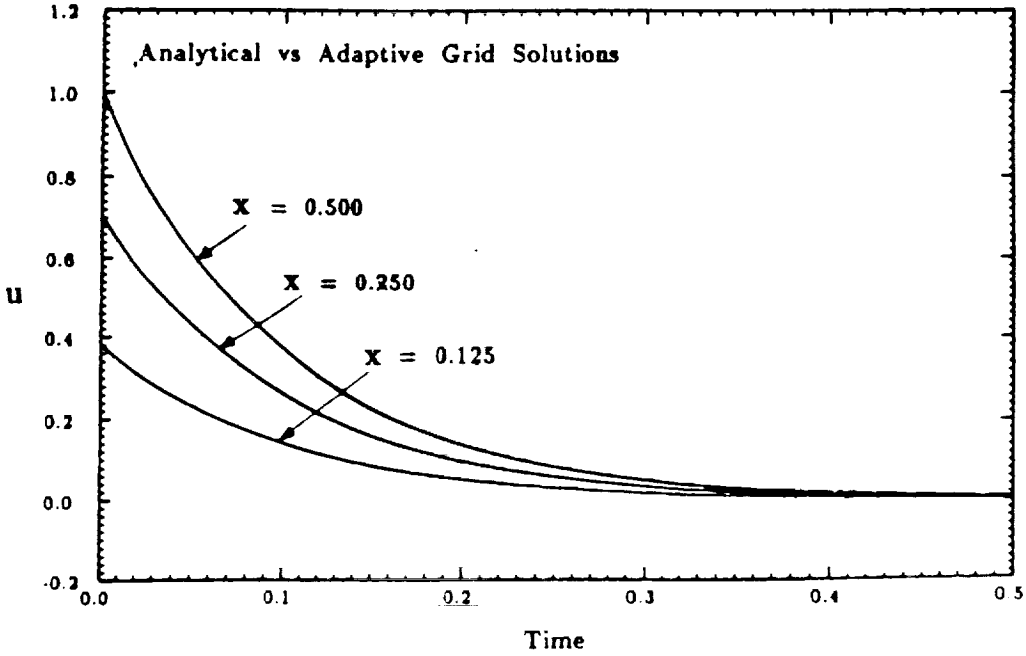

Figure 5.2.2: Adaptive Grid Solution to Problem #2; Analytical and Adaptive Grid Solutions Overlie

Table 5.2.1: Test Problem #2, Multigrid Results

α	δ	η	total WU	average WU per time step	CPU time (seconds)
0.25	0.220	0.625	2404	4.81	8.50
0.25	0.125	0.500	2212	4.42	7.82
0.25	0.125	0.700	2409	4.82	8.37
0.25	0.100	0.500	2212	4.42	7.87
0.25	0.100	0.450	2212	4.42	7.37

and CPU time. Varying δ and η had little effect on the overall efficiency of the algorithm, yet the number of work units needed to solve the problem for a given time step differed until the solution began to approach steady state (see Figures 5.2.3 and 5.2.4).

In the first case (Figure 5.2.3), a pulse in the number of work units needed to obtain the solution (for a particular time step) appears just prior to a steep 'drop off'. The drop off in the amount of work units needed is also present in the second case, Figure 5.2.4 . To explain the occurrence of these effects, one should consider both the solution process or algorithm as well as the actual solution to the problem being solved. With this in mind, as the solution approaches steady state, it varies less and less with each new time step and consequently, the initial estimate of the solution for each new time step becomes more accurate. Also, as steady state is approached, the magnitude of the low frequency errors, which dominate this problem, decreases. When this is coupled with multigrid process, the drop off and pulse shown in Figures 5.2.3 and 5.2.4 is produced. So, as the solution process progresses in time, the need for additional coarse grid work (to smooth out low frequency errors) is eliminated and the point where this occurs shows up as a sudden drop off in the number of work units required to solve this problem for the particular time step. Prior to the drop off, the solution process goes from the coarsest to finest grid level and then returns to the coarser grids before finishing off on

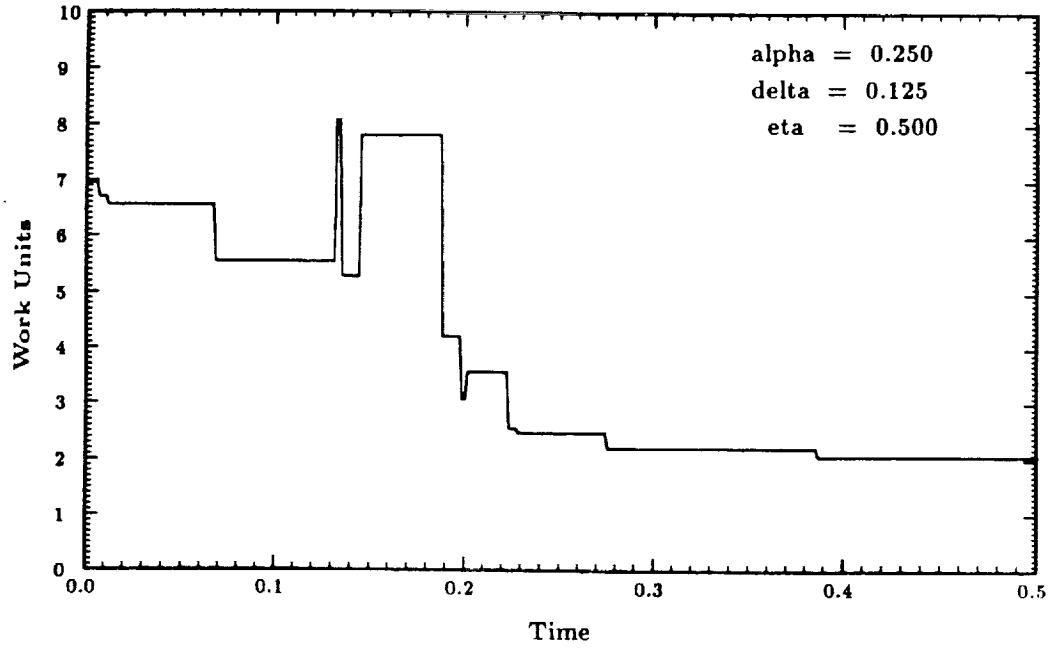

Figure 5.2.3: Work - Case #1

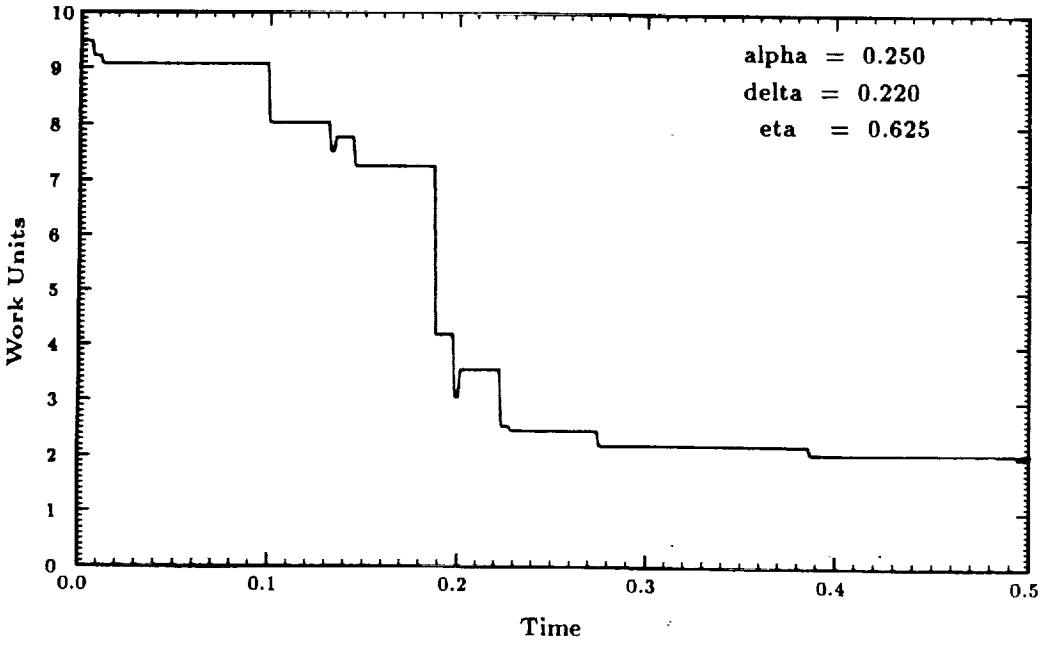

Figure 5.2.4: Work - Case #2

the finest level. After the drop off, the solution process stops upon reaching the finest level and then continues on to the next time step.

The pulse (Figure 5.2.3) is produced in a somewhat similar manner. Initially, the amount of low frequency error present in the estimate of the solution leads to inefficient relaxation sweeps. This results in the algorithm cycling back to the coarser grids (before reaching the finest level) to smooth out low frequency errors. Then, upon reaching the finest level for the first time, the solution converges to the desired accuracy and the algorithm proceeds to solve the problem for the next time step. Later, as the solution process approaches steady state, the amount of low frequency errors decrease and results in a delay of the onset of inefficient relaxation sweeps until the finest level is reached. At this point, execution returns to the coarser grids. Delaying the return to coarser levels in this manner means that some of the additional work is performed on the finer grids. It is this additional, and relatively costly, computational work that produces the pulse.

Comparing the multigrid algorithm to both the Gauss-Seidel and tridiagonal methods yields results similar to those for test problems #1. The Gauss-Seidel algorithm requires 54807 iterations in 116.6 CPU seconds to determine the solution to near steady state ($t = 0.50$). Based on work units, the multigrid method solves this problem 24.7 times faster than the Gauss-Seidel algorithm, but is only 14.8 times faster when CPU time is compared. The

direct solver solves the problem out to steady state in 1.74 CPU seconds, or 4.54 times quicker than the multigrid method does.

The adaptive grid method solves the problem to steady state in 5.31 CPU seconds and requires an average of 0.762 WU per time step. To compare the performance of the adaptive grid and multigrid algorithms, both the number of work units and the amount of CPU time needed to obtain the solution is considered. Based on work units, the adaptive grid scheme is about 5.8 times faster than the multigrid method. While this appears to be a significant improvement, this comparison may be misleading as it does not account for the overhead (interpolations, bookkeeping, etc.) required by both methods. By comparing CPU time, one finds that the adaptive grid method solves the problem approximately 1.5 times faster. While the adaptive grid method outperforms the multigrid program, it is still about 3 times slower than the direct solver.

5.3 Results for Test Problem #3

Test problem #3 is the one-dimensional convection-diffusion equation given by (4.3.1). The problem is solved using a multigrid method employing the use of an upwind/downwind Gauss-Seidel relaxation scheme. For comparative purposes the problem is solved with a direct solver. In addition, the adaptive grid method is also used to obtain the solution. In order to solve test problem #3, the domain is discretized onto a 225 node grid. For the multigrid and adaptive grid methods, 8 grid levels are used with the coarsest and finest grids containing 6 and 225 nodes respectively. The solution is found up to near steady state ($t = 0.20$) using 200 time steps. To solve test problem #3, a fixed time increment of $\Delta t = 0.001$ is used in all the programs.

The multigrid and analytical solutions are shown in Figures 5.3.1 and 5.3.2. In Figure 5.3.1, the solution is presented as it stands at a time of 0.05. Figure 5.3.2 shows how the solution at certain nodes ($x = 0.25,\ 0.50$, and 0.75) vary with time up to the onset of steady state. The multigrid program with the switching parameters set at $\alpha = 0.25$, $\delta = 0.22$, and $\eta = 0.70$ is used to solve this problem. The solution up to near steady state is found in 4.47 CPU seconds and requires an average of 15.5 WU per time step. Again, the values of δ and η were varied in order to see how they affect program efficiency (see Table 5.3.1). For this problem, both the CPU time required to find the solution and the average number of work units needed per time step remained

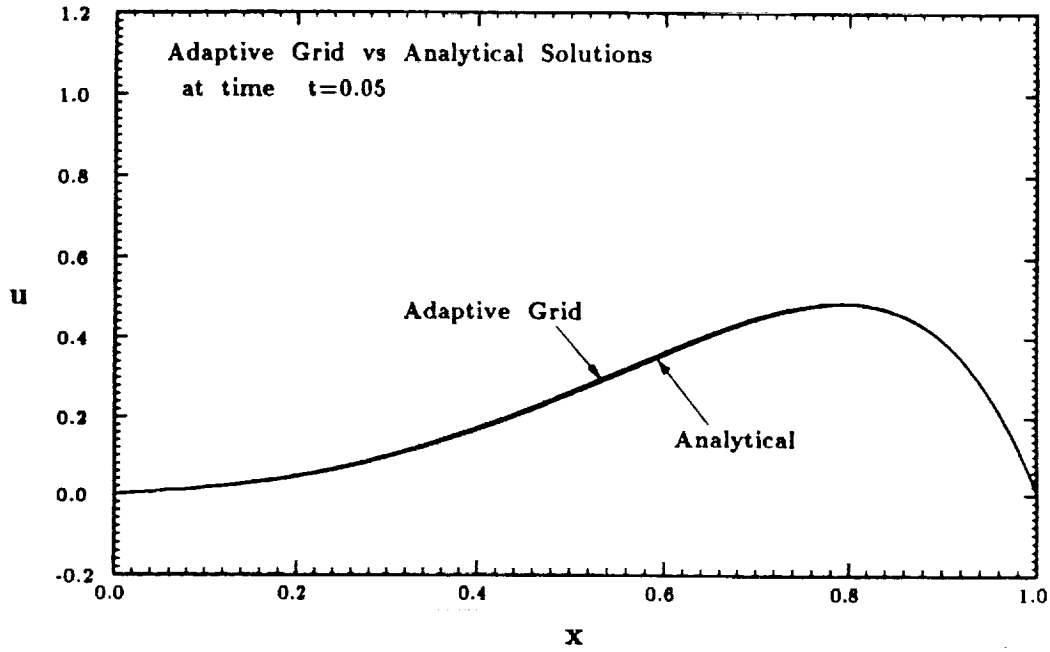

Figure 5.3.1: Adaptive Grid Solution to Problem #3, Snapshot View

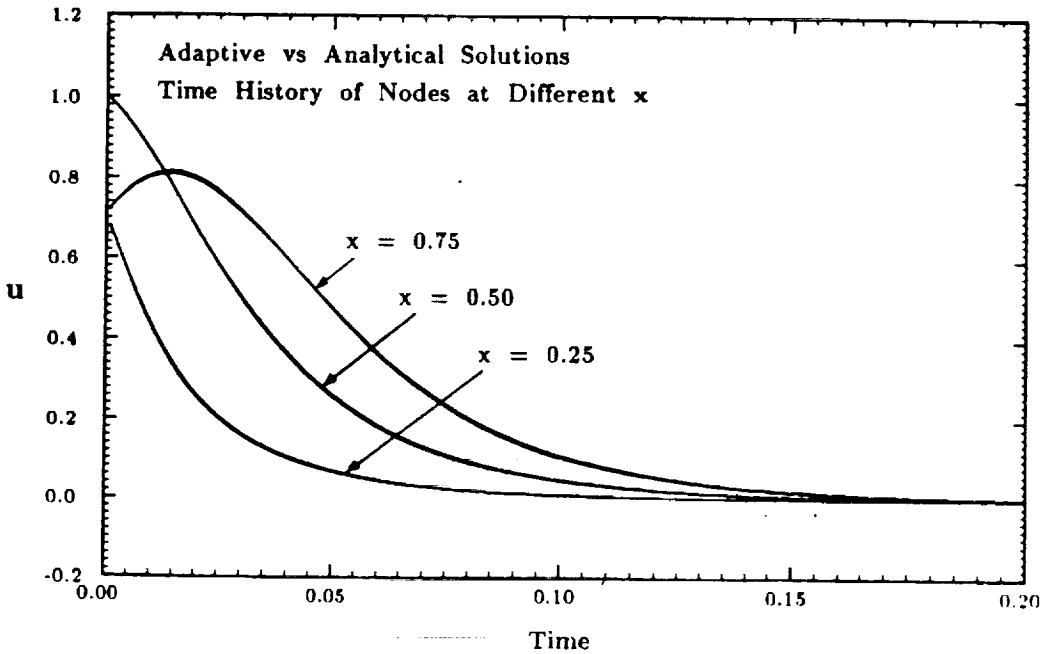

Figure 5.3.2: Adaptive Grid Solution to Problem #3, Time Histories; Analytical and Multigrid Solutions Overlie

Table 5.3.1: Test Problem #3, Results; Time = 0.050

α	δ	η	total WU	average WU per time step	CPU time (seconds)
0.25	0.125	0.500	879	17.6	5.31
0.25	0.220	0.625	821	16.4	4.79
0.25	0.220	0.700	773	15.5	4.47
0.25	0.220	0.800	920	18.4	4.93
0.25	0.220	0.500	872	17.4	5.35
0.25	0.220	0.400	859	17.2	5.36
0.25	0.100	0.700	828	16.6	4.84
0.25	0.300	0.700	815	16.3	4.72
0.25	0.500	0.700	1114	22.3	6.59
0.25	0.700	0.700	2433	48.6	14.86

relatively constant with respect to δ and η.

The adaptive grid solution to test problem #3 is compared to the analytical solution in Figures 5.3.3 and 5.3.4 . In Figure 5.3.3, the solution is depicted as it exists at a time of $t = 0.05$, while Figure 5.3.4 shows how the solution at selected nodes progresses through time. The adaptive grid scheme requires 3.87 CPU seconds to find the solution up to steady state and averages 1.06 WU per time increment.

In contrast to the results obtained from the multigrid and adaptive grid algorithms, the direct solution (tridiagonal) method solves test problem #3 in 1.19 CPU seconds. Thus, the direct solver outperforms the multigrid method again, but this time by about a factor of 13.7 times. The adaptive grid method fared a bit better, as it ran about 3.5 times slower than the tridiagonal scheme and about 4.2 times quicker than the multigrid algorithm. In further comparing the multigrid and adaptive grid schemes, the multigrid method required 13.11 times more work units to determine the solution up to near steady state.

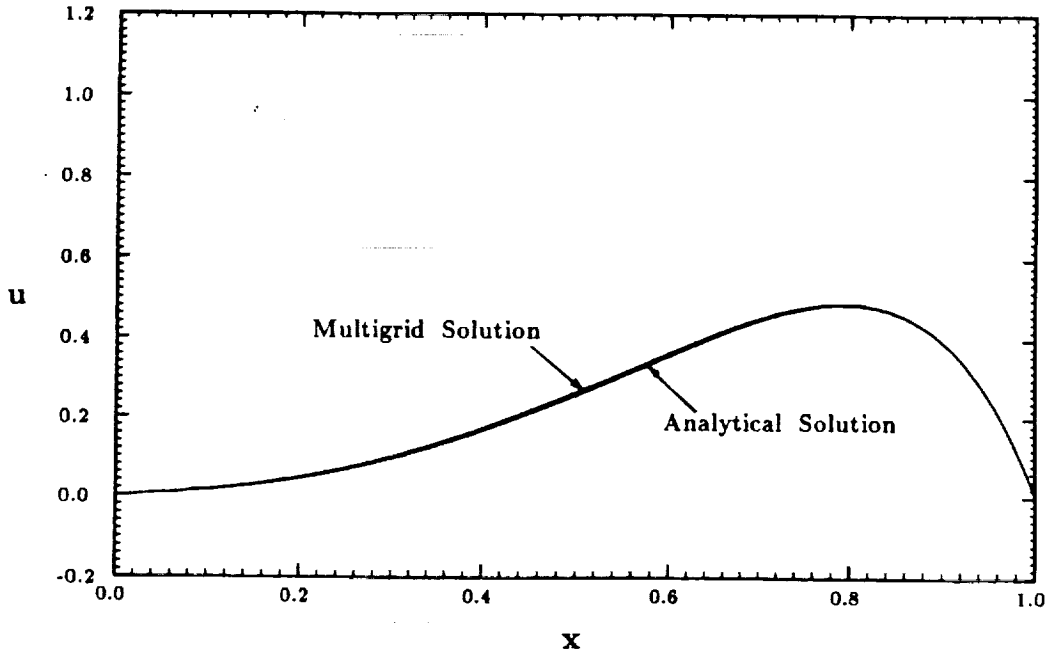

Figure 5.3.3: Multigrid Solution to Problem #3, Snapshot View

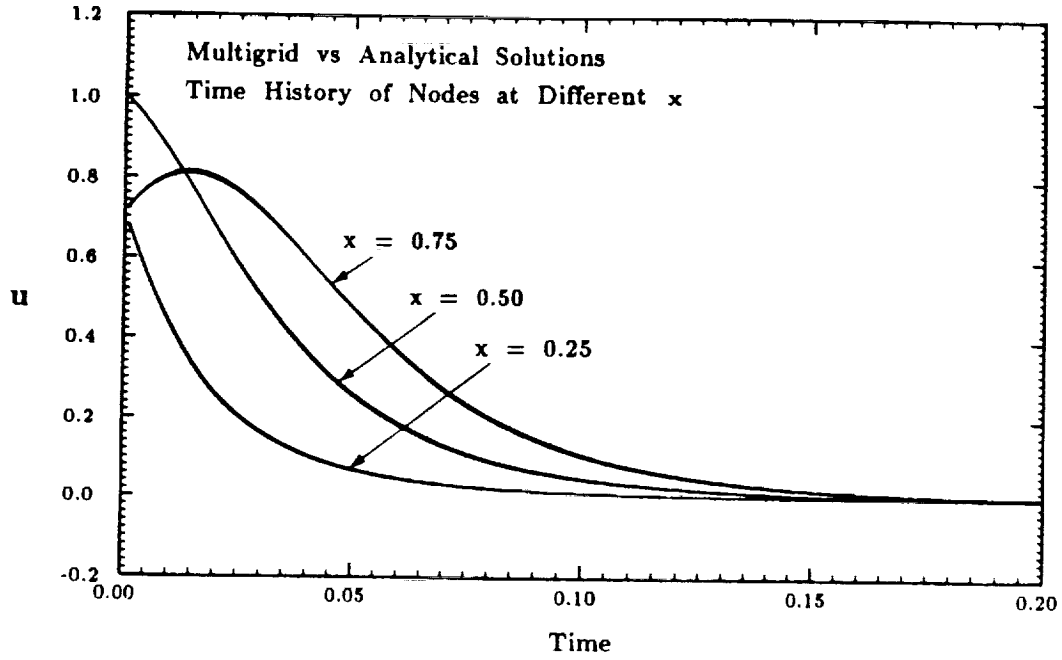

Figure 5.3.4: Multigrid Solution to Problem #3, Time Histories; Analytical and Multigrid Solutions Overlie

5.4 Results for Test Problem #4

Test problem #4 uses the nonlinear viscous Burgers' equation given by (4.4.1) and (4.4.2). This problem is solved using both an iterative tridiagonal and adaptive grid algorithms. As the analytical solution to test problem #4 was not found, the adaptive grid algorithm is verified by comparing its solution to that found using the tridiagonal method. The Burgers' equation is discretized onto a 1025 node grid. The adaptive grid program uses 8 levels of discretization with the coarsest level containing 9 nodes. The solution is found up to $t = 1.00$ using 100 time steps with the constant time increment $\Delta t = 0.01$. The problem is also solved using larger time increments in order to determine the largest time step which will yield a satisfactory solution.

The tridiagonal program, used to generate the data presented here, utilizes the adaptive grid residual norm (for the analogous adaptive grid case) from the final time increment as the convergence criteria for each time step. So, the convergence criteria remains the same from one time increment to another. Additionally, the tridiagonal program was modified such that the adaptive grid residual norm from each time step (found from the finest grid level) is used as the convergence criteria for the corresponding time increments. This alteration in the tridiagonal program did not result in any appreciable changes in the performance of the program for this problem.

The solutions from the tridiagonal and adaptive grid programs are presented in Figure 5.4.1 . The adaptive grid method solves test problem #4 in approximately 9.32 CPU seconds and requires an average of 0.661 WU per time step. In contrast, the direct solver (which employs Picard iterations to handle the nonlinear terms) finds the solution in 30.37 CPU seconds and requires an average of 7.04 work units (iterations) per time step. So, for this problem, the adaptive grid program is about 3.26 times faster than the tridiagonal method. Comparing computational work shows that the tridiagonal method requires about 10.66 times more work units than the nonlinear adaptive grid algorithm.

The nonlinear adaptive grid program parameters, *toler* and ϵ, are varied in order to see what affect they have on program flow and the resulting solution. For this problem, variations in *toler* (the criteria applied to the change in the residual norm between iterations ($\Delta e = e^k - \bar{e}^k$) to determine whether additional Picard iterations are required) have almost no effect on the solution and solution process (see Table 5.4.1). This occurs because the solution rapidly converges within each subgrid, usually within 2 or 3 iterations. With the exception of the coarsest grid, the residual norms, calculated within each subgrid, become constant ($\Delta e = 0$) in 2 iterations. Thus, varying *toler* has little effect as it is designed to signal when the residual norms become constant. Since the residual from the adaptive grid program is used as the convergence

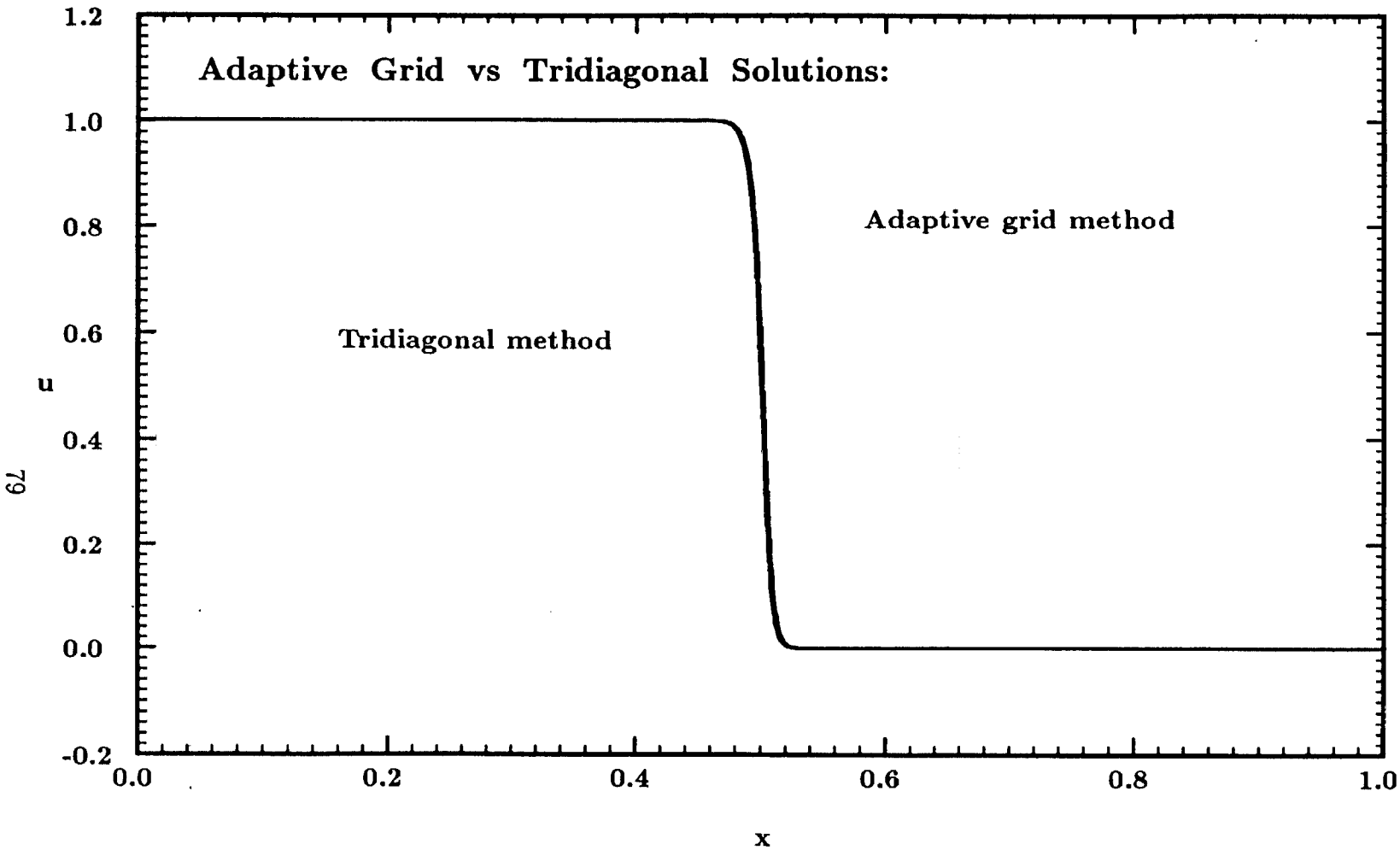

Figure 5.4.1: Adaptive Grid Solution to Problem #4, Burgers' Equation

Table 5.4.1: Test Problem #4 Results, Variations in *toler*

where $\log(\epsilon) = -8$, and $\Delta t = 0.01$

adaptive subgrids use 2 converged nodes per boundary

log toler	AD average WU per time step	TRID average WU per time step	AD CPU time sec	TRID CPU time sec	AD residual $\|F - Lu\|$	TRID residual $\|F - Lu\|$
-2	.643	7.04	9.35	30.37	3.81E-4	2.26E-4
-3	.647	7.04	9.28	30.37	3.81E-4	2.26E-4
-4	.652	7.04	9.32	30.37	3.81E-4	2.26E-4
-5	.657	7.04	9.30	30.37	3.81E-4	2.26E-4
-6	.663	7.04	9.28	30.37	3.81E-4	2.26E-4
-7	.669	7.04	9.39	30.37	3.81E-4	2.26E-4
-8	.675	7.04	9.34	30.37	3.81E-4	2.26E-4
-9	.681	7.04	9.34	30.37	3.81E-4	2.26E-4

criteria for the direct solver, the residuals found by the tridiagonal method remain constant for each of the cases presented in Figure 5.4.1.

The parameter ϵ is the convergence criteria (applied to the relative truncation error) used to construct the adaptive subgrids. The more stringent (smaller) ϵ becomes, the more the adaptive grid algorithm strives to improve the accuracy of the solution by adding finer grids. As ϵ is decreased, the algorithm attempts to improve the accuracy of the solution at the expense of increased computational work (see Figures 5.4.2 and 5.4.3). If the value of ϵ is too large, not enough computational work is invested in order to adequately resolve the location of the front present in the solution. This results in a solution in which the front lags behind its actual location. As depicted in Figure 5.4.4, the larger ϵ is, the more the front lags. The data collected while varying ϵ is presented in Tables 5.4.2a and 5.4.2b .

The size of the time increments used are important. Using a time increment of 0.01 results in a solution with a well defined front. Increasing the size of the time steps, reduces the CPU time needed to solve the problem, but will result in some diffusion being present in the front. Using still larger time increments, adds an increasing amount of diffusion as shown in Figure 5.4.5 .

Time increments smaller than 0.01 may be used to obtain a slightly more accurate approximation of the solution, but will require a great deal more computation time (see Figure 5.4.6 and Table 5.4.3). The largest time incre-

ment yielding a satisfactory solution is $\Delta t = 0.01$. For this case the adaptive grid algorithm is 3.26 times faster than the tridiagonal method.

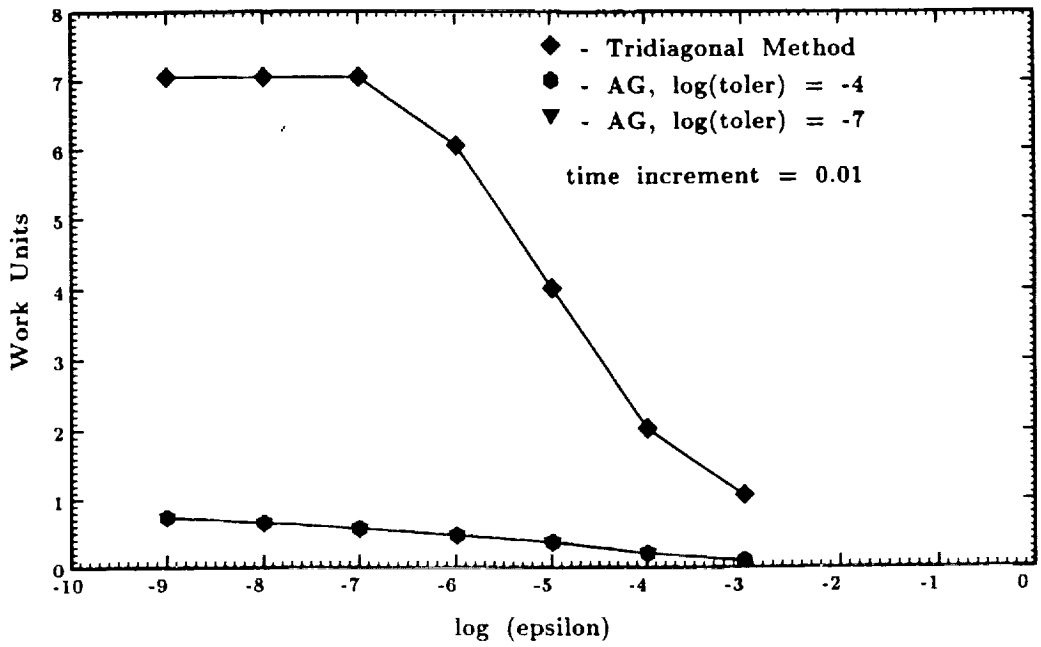

Figure 5.4.2: Work vs. Log(ϵ), Burgers' Equation

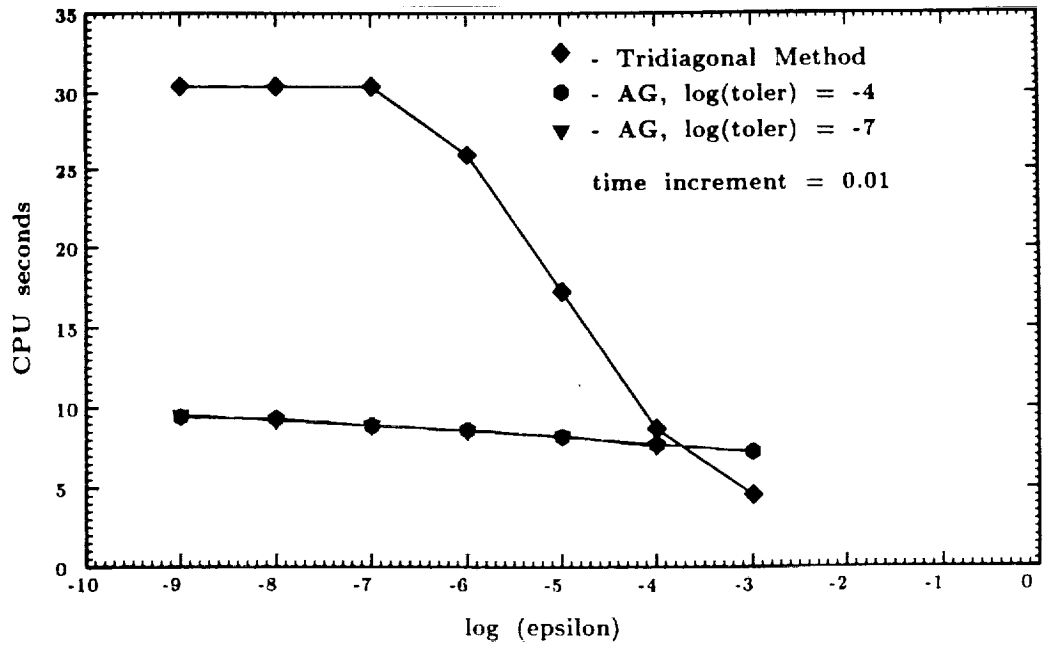

Figure 5.4.3: CPU Time vs. Log(ϵ), Burgers' Equation

C-2

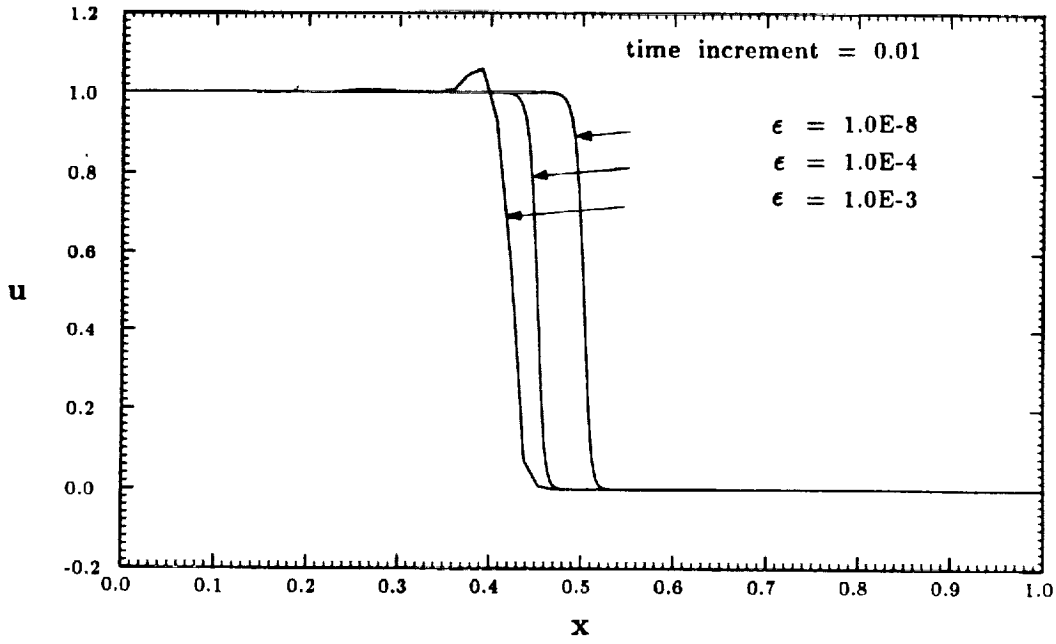

Figure 5.4.4: Effects on the Solution Caused by Varying ϵ

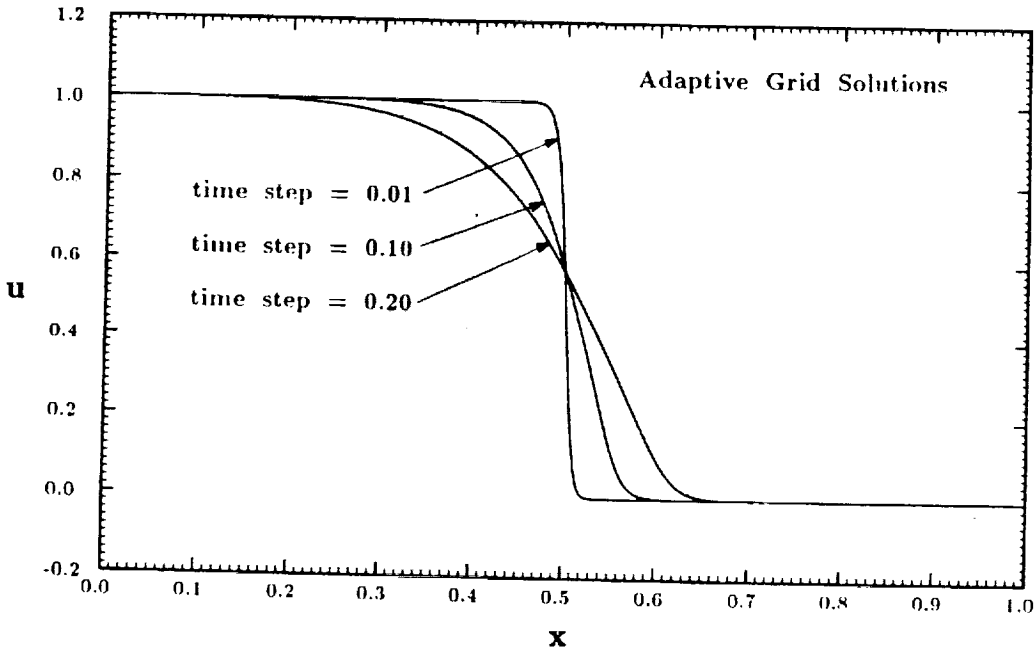

Figure 5.4.5: Diffusion of Front Due to Large Time Increments

Table 5.4.2a: Test Problem #4 Results, Variations in ϵ

where $\log(toler) = -4$, and $\Delta t = 0.01$

log ϵ	AD average WU per time step	TRID average WU per time step	AD CPU time sec	TRID CPU time sec	AD residual $\|F - Lu\|$	TRID residual $\|F - Lu\|$
-9	.732	7.04	9.42	30.37	3.81E-4	2.26E-4
-8	.652	7.04	9.32	30.37	3.81E-4	2.26E-4
-7	.570	7.04	8.88	30.37	4.08E-4	2.26E-4
-6	.475	6.05	8.62	25.90	1.21E-3	8.34E-4
-5*	.369	4.01	8.19	17.21	3.11E-2	1.07E-3
-4*	.208	2.02	7.71	8.68	0.2097	0.1289
-3*	.112	1.07	7.27	4.56	0.4192	0.4165

Table 5.4.2b: Test Problem #4 Results, More Variations in ϵ

where $\log(toler) = -7$, and $\Delta t = 0.01$

log ϵ	AD average WU per time step	TRID average WU per time step	AD CPU time sec	TRID CPU time sec	AD residual $\|F - Lu\|$	TRID residual $\|F - Lu\|$
-9	.732	7.04	9.52	30.37	3.81E-4	2.26E-4
-8	.652	7.04	9.22	30.37	3.81E-4	2.26E-4
-7	.570	7.04	8.90	30.37	4.08E-4	2.26E-4
-6	.475	6.05	8.57	25.90	1.21E-3	8.34E-4
-5*	.369	4.01	8.24	17.21	3.11E-2	1.07E-3
-4*	.208	2.02	7.59	8.68	0.2097	0.1289

* front lags

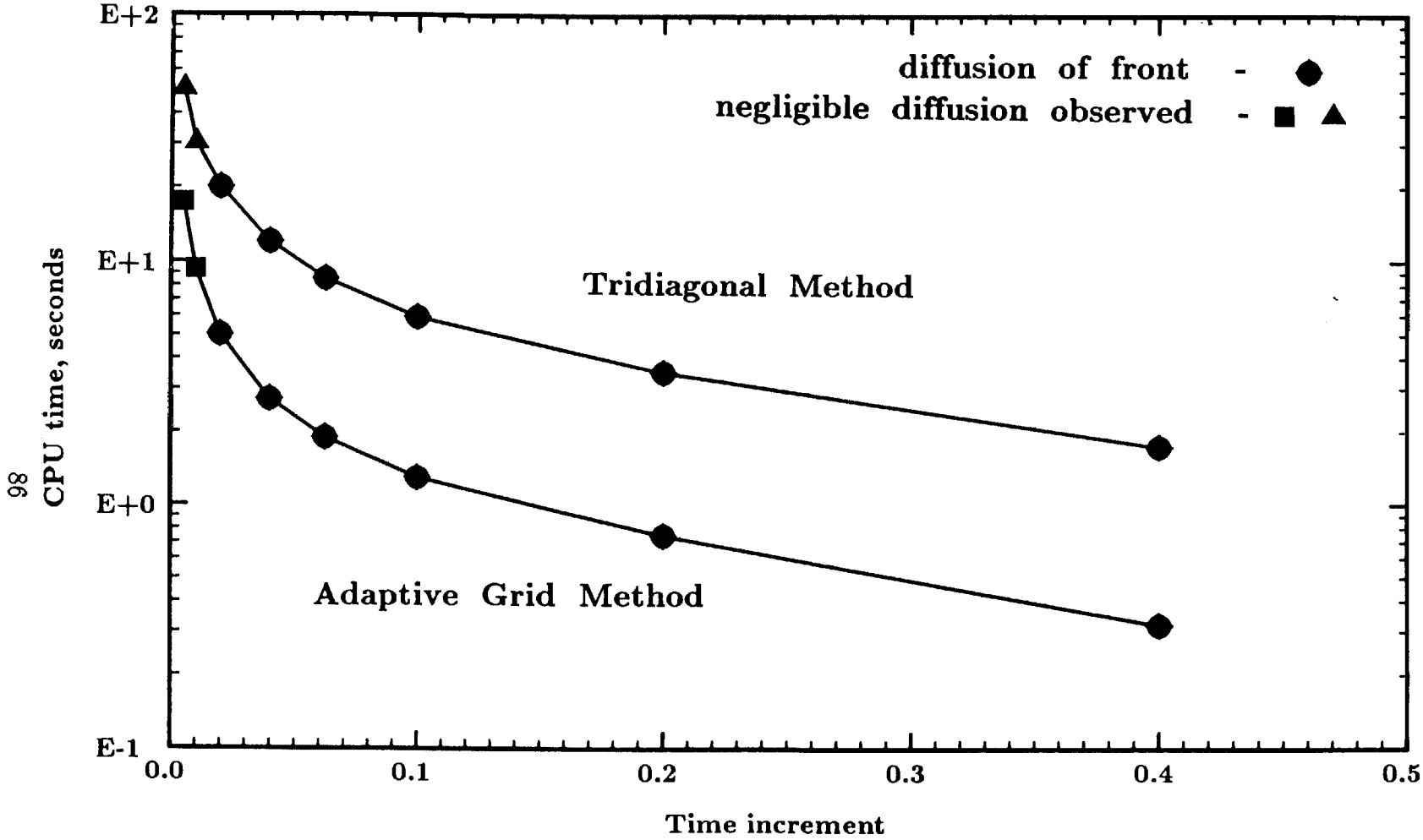

Figure 5.4.6: CPU Time vs. Time Increment, Burgers' Equation

Table 5.4.3: Test Problem #4 Results, Time Increment Varied

where $\log(\epsilon) = -8$

Δt sec	AD average WU per time step	TRID average WU per time step	AD CPU time sec	TRID CPU time sec	AD residual $\|F - Lu\|$	TRID residual $\|F - Lu\|$
0.005	0.549	6.02	17.47	50.31	2.89E-4	9.15E-5
0.01	0.652	7.04	9.32	30.37	3.81E-4	2.26E-4
0.02*	0.822	9.04	5.01	20.02	3.62E-4	1.43E-4
0.04*	1.09	10.12	2.72	12.06	2.80E-4	2.66E-4
0.0625*	1.34	11.19	1.90	8.56	2.68E-4	2.46E-4
0.10*	1.61	12.30	1.30	5.94	4.36E-4	2.37E-4
0.20*	2.02	14.40	0.75	3.52	9.63E-4	1.42E-4
0.40*	2.03	14.00	0.32	1.74	9.30E-2	1.57E-4

* diffusion of front

5.5 Results for Test Problem #5

The fifth test problem is the nonlinear Richards' equation (presented in Chapter 4.5) modelling one-dimensional water flow in an unsaturated soil. This problem is solved using an adaptive grid algorithm and the tridiagonal method, each of which employs the use of Picard iterations to handle the nonlinearities. As with the previous problem, the adaptive grid and tridiagonal solutions are compared with each other, since the analytical solution to the problem is not known. To get an equitable comparison between the two methods, the Picard iterations for the tridiagonal scheme are continued until the residual is the same as that found by the adaptive grid program. For the adaptive grid scheme, the problem is discretized such that the finest grid contains either 481 nodes, or 641 nodes. The total number of grid levels used, as well as the number of nodes on the coarsest grid, are varied in order to see how they affect the solution and computation time. The tridiagonal method solves this problem on uniform grids consisting of 481 nodes, and 641 nodes. Both algorithms solve the problem up to times of 5, 15, and 35 days. The resulting solutions are plotted in Figure 5.5.1 . Additionally, various time increments are used in solving test problem #5, up to times of 5, 15, and 35 days.

A problem arising with the nonlinear tridiagonal algorithm is one in which the low frequency errors present in the solution cause the relaxation sweeps (iterations) to become very inefficient. This leads to a situation where the

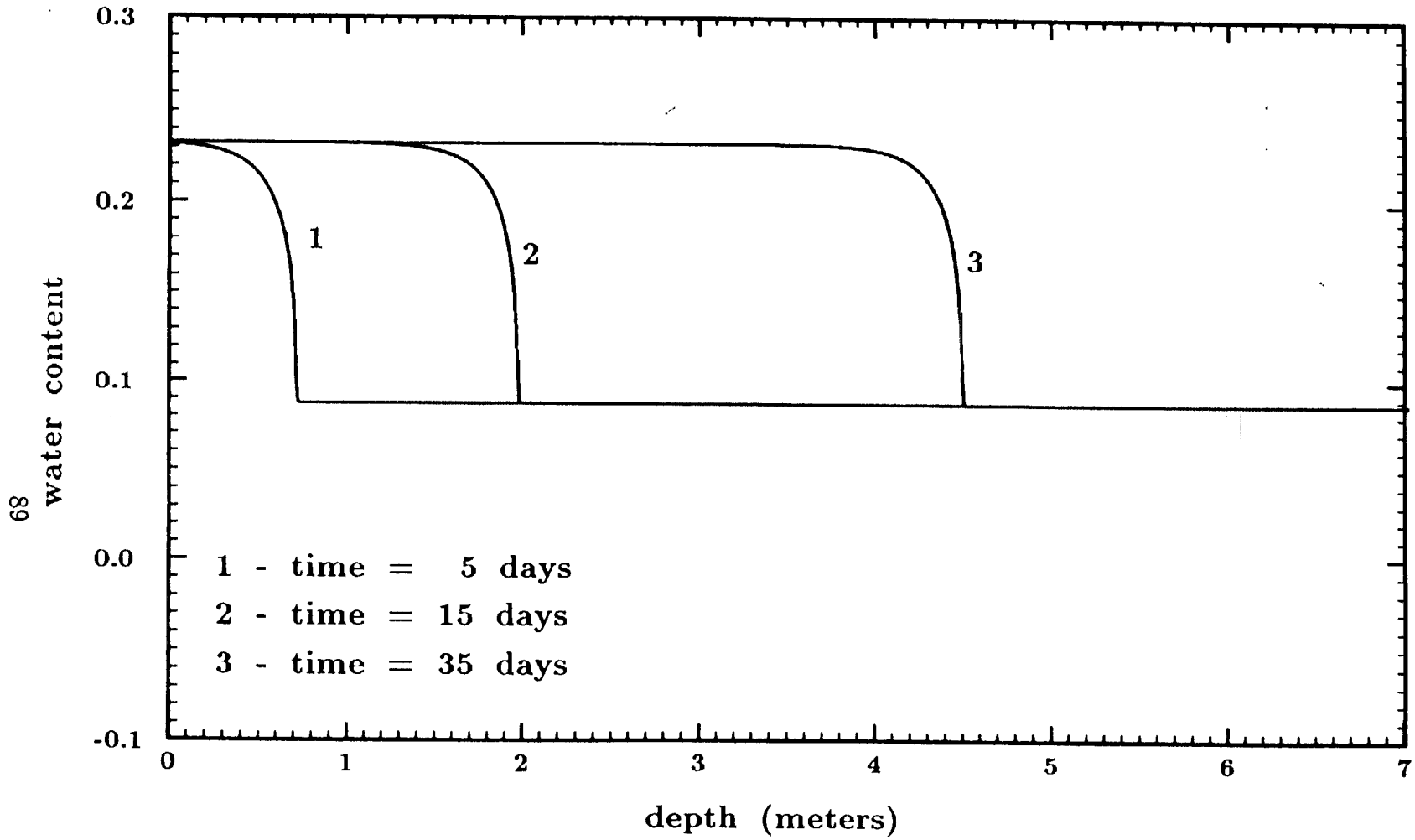

Figure 5.5.1: Adaptive Grid vs. Tridiagonal Solution to Problem #5, Richards' Equation

Adaptive Grid and Tridiagonal Solutions Overlie

residual remains almost constant between iterations, and may even converge on a value greater than the one desired, resulting in an infinite loop. To detect such an occurrence, the parameter 'tolerce' is introduced into the tridiagonal algorithm and compared against the change (since the previous iteration) in the residual norm Δe. As long as $\Delta e > tolerce$, the algorithm proceeds as usual. When $\Delta e < tolerce$, the Picard iterations are halted (as if the residual had converged to within the desired accuracy) and the tridiagonal algorithm proceeds to solve for the next time step. Doing this, avoids creating an infinite loop, but tends to 'lock in' a low frequency error into the solution, which consequently gets propagated onto the remaining time steps.

The water infiltration problem is solved up to a time of 5 days, using a time increment of 0.10 day, with both the adaptive grid and tridiagonal algorithms. Performing a mass balance shows a gain in mass of 0.104 cm H_2O (about 1.15 percent of the total mass within the domain) in the solutions found by each algorithm. The tridiagonal method solved the problem on a 481 node grid in 309.95 CPU seconds. The adaptive grid program solves the problem using several grid levels, such that the finest grid contains 481 nodes. The program was run several times, as the number of grid levels, and convergence criteria (ϵ) were varied. For these cases, the adaptive grid method solves the problem anywhere from 66.14 to 142.02 CPU seconds, depending upon the value of ϵ and the number of grid levels used (see Table 5.5.1).

Table 5.5.1: Test Problem #5, Results ; 481 Fine Node Grid*

where $\log(toler) = -5$, time = 5 days, and $\Delta t = 0.10$

# of grid levels	# nodes coarsest grid	AG CPU time sec	TRID CPU time sec	AG residual $\|F - Lu\|$	TRID residual $\|F - Lu\|$	log ϵ
3	121	138.92	309.95	5.614E-5	1.615E-4	-7
4	61	93.11	309.95	5.614E-5	1.615E-4	-7
5	31	73.97	309.95	5.614E-5	1.615E-4	-7
6	16	66.14	309.95	5.614E-5	1.615E-4	-7
3	121	141.05	309.95	5.614E-5	1.615E-4	-8
4	61	93.50	309.95	5.614E-5	1.615E-4	-8
5	31	74.48	309.95	5.614E-5	1.615E-4	-8
6	16	68.16	309.95	5.614E-5	1.615E-4	-8
3	121	142.04	309.95	5.614E-5	1.615E-4	-9
4	61	95.84	309.95	5.614E-5	1.615E-4	-9
5	31	75.01	309.95	5.614E-5	1.615E-4	-9
6	16	67.21	309.95	5.614E-5	1.615E-4	-9

* tridiagonal and adaptive grid algorithms each yield mass gains of 0.104 cm H_2O

In each case, the adaptive grid method obtained a residual value of 5.614×10^{-5}. The tridiagonal scheme attempted to obtain a solution of equal accuracy (by using the residual value to obtained by the adaptive grid method as the convergence criteria), but only managed to get a residual of 1.615×10^{-4}. This occurred because low frequency errors in the solution led to very inefficient relaxation sweeps which converged on a residual value greater than the one sought. So for this case, the adaptive grid algorithm outperformed the tridiagonal scheme, solving the problem 2.23 to 4.69 times faster, while obtaining a residual value approximately half an order of magnitude less than that from the tridiagonal method.

The two algorithms are also used to solve the water infiltration problem discretized onto a grid (or finest grid) with 641 nodes, up to a time of 5 days, using a time increment of 0.10 day. In finding the solution, both algorithms produce a mass gain of 0.078 cm H_2O or about 0.86 percent of the total mass. The program using the tridiagonal method solved this problem in 379.26 CPU seconds with a residual value of 1.365×10^{-4}. The adaptive grid method obtains the solution in 80.50 to 194.77 CPU seconds (about 1.95 to 4.71 times faster than the tridiagonal method), while obtaining a residual value of 5.732×10^{-5} (see Table 5.5.2).

Table 5.5.2: Test Problem #5, Results ; 641 Fine Node Grid*

where $\log(toler) = -5$, time = 5 days, and $\Delta t = 0.10$

# of grid levels	# nodes coarsest grid	AG CPU time sec	TRID CPU time sec	AG residual $\|F - Lu\|$	TRID residual $\|F - Lu\|$	log ϵ
3	161	194.77	379.26	5.732E-5	1.365E-4	-6
4	81	124.87	379.26	5.732E-5	1.365E-4	-6
5	41	111.88	379.26	5.732E-5	1.365E-4	-6
6	21	89.14	379.26	5.732E-5	1.365E-4	-6
7	11	80.50**	379.26	5.732E-5	1.365E-4	-6
3	161	190.99	379.26	5.732E-5	1.365E-4	-8
4	81	125.51	379.26	5.732E-5	1.365E-4	-8
5	41	98.52	379.26	5.732E-5	1.365E-4	-8
6	21	87.76	379.26	5.732E-5	1.365E-4	-8
7	11	83.21	379.26	5.732E-5	1.365E-4	-8
3	161	190.30	379.26	5.732E-5	1.365E-4	-9
4	81	125.82	379.26	5.732E-5	1.365E-4	-9
5	41	98.96	379.26	5.732E-5	1.365E-4	-9
6	21	87.68	379.26	5.732E-5	1.365E-4	-9
7	11	83.78	379.26	5.732E-5	1.365E-4	-9

* tridiagonal and adaptive grid algorithms each yield mass gains of 0.0784 cm H_2O

** adaptive grid mass loss = 0.0112 cm H_2O

For the adaptive grid algorithm, the number of grid levels and the convergence criteria are varied in order to see what effect they have on the solution and computation time. These cases are presented in Table 5.5.1 and Table 5.5.2 . Increasing the number of grid levels results in a decrease of the computation time required to solve the problem, yet it has no effect on the residual norm. Variations in the convergence criteria also have no effect on the residual norm. Additionally, the time needed to solve the problem remains approximately constant as the convergence criteria is varied. For example using 6 grid levels and a finest grid of 641 nodes, the problem is solved in 89.14 seconds with $\epsilon = 10^{-6}$. Decreasing the convergence criteria to $\epsilon = 10^{-8}$ results in a requirement of 87.76 CPU seconds to solve the problem, a further reduction to $\epsilon = 10^{-9}$ results in a time requirement of 87.68 CPU seconds.

In solving problem #5 out to 15 days using $\Delta t = 0.10$ day, the performance of the adaptive grid method degrades as more computation time is required per time step, but it is still the faster of the two. The degradation in performance worsens as the problem is solved up to a time of 35 days while using the same time increment. With the adaptive grid algorithm, the adaptive subdomains appear near the front and over those portions of the domain behind the front. The extent of the subgrids used to solve the problem at a time of 15 days is shown in Figure 5.5.2. As the solution progresses in time, the location of the front penetrates deeper and deeper into the domain,

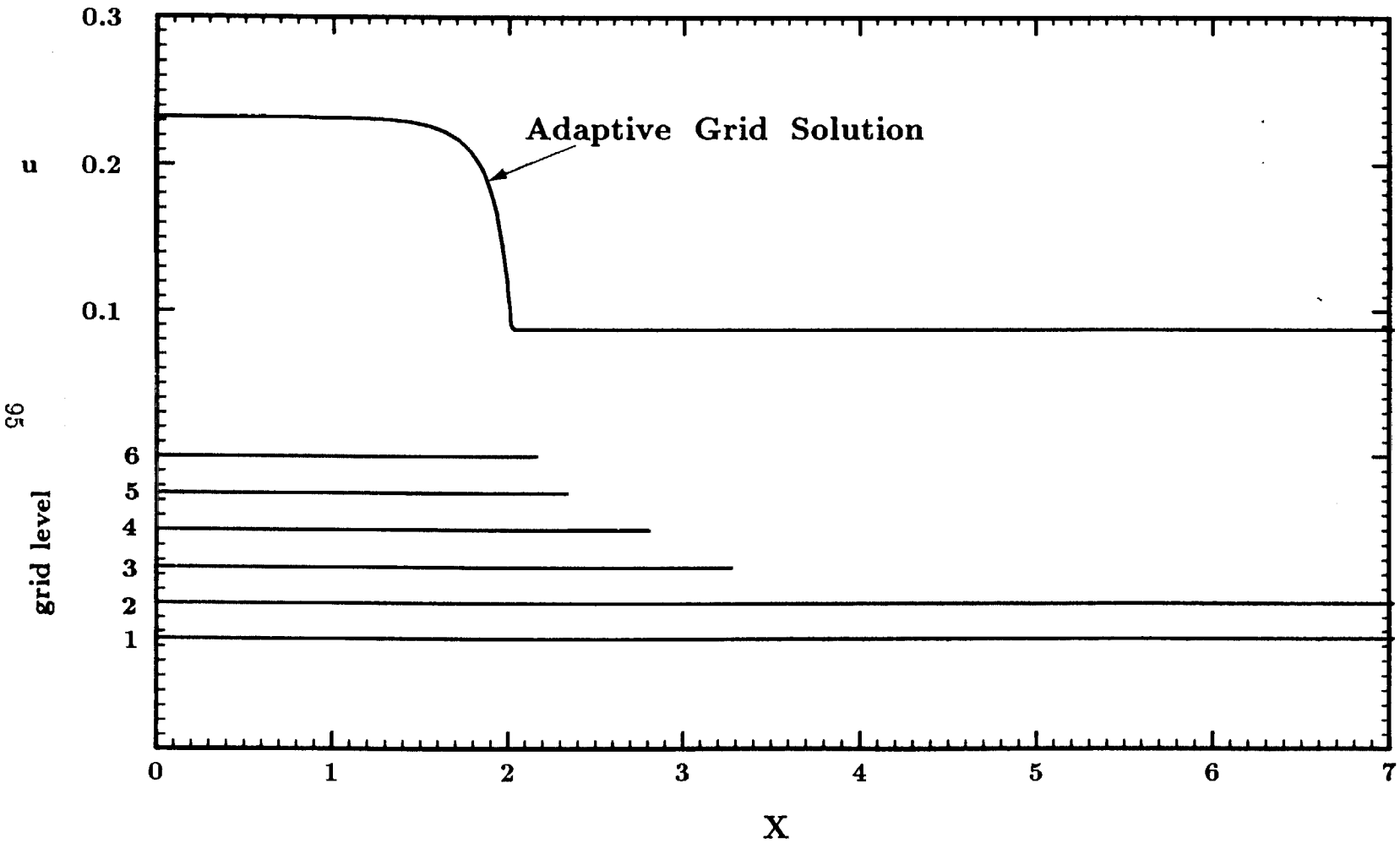

Figure 5.5.2: Adaptive Grids Used at Time = 15 Days

and consequently allows for the subdomains to cover larger portions of the domain, resulting in a slowing of the algorithm. For example, problem #5 is solved on a 481 node grid, using a time increment of 0.20 day. The adaptive grid method obtains a solution for a time of 5 days, 5.05 times faster than the tridiagonal scheme. In solving the problem out to 15 days, the adaptive grid method is 3.57 times faster. And finally in solving the problem out to 35 days, the performance of the adaptive grid degrades even further as it is only 2.60 times faster than the tridiagonal scheme.

The Richards' equation is solved (up to times of 5, 15, and 35 days) several times with both programs as the time increments are increased with each new set of computer runs. The data collected are presented in Tables 5.5.3, 5.5.4, and 5.5.5. As the time steps are increased, errors in the mass conservation slowly rise. Additionally, oscillations in the residual error $(F - Lu)$ appear with the use of the larger time increments (see Figures 5.5.3, 5.5.4, and 5.5.5). The spikes appearing in Figures 5.5.3 and 5.5.4 occur at most subdomain boundaries. The magnitude of the spikes generally decreases as smaller time steps are used. These spikes occur since Dirichlet conditions are applied to the subdomain boundaries (which are in the interior of the domain and do not correspond to the actual boundaries of the problem), and so, the flux across these boundaries is not considered in the current numerical model. This leads to the introduction of the errors appearing at the subdomain boundaries.

Table 5.5.3: Test Problem #5, Results for Time = 5 days

where $\log(\epsilon_x) = -8$ and $\log(toler) = -5$

AG residual $\|F - Lu\|$	TRID residual $\|F - Lu\|$	AG CPU time sec	TRID CPU time sec	AG mass gain cm H_2O	TRID mass gain cm H_2O	Δt day
for 6 grid levels and 481 nodes on finest grid						
1.05E-5	3.50E-5	254.0	751.8	8.66E-3	0.1040	0.02
2.80E-5	8.50E-5	116.9	403.6	0.1041	0.1041	0.05
5.61E-5	1.62E-4	67.7	310.0	0.1042	0.1042	0.10
1.25E-4	2.95E-4	41.0	206.9	0.1045	0.1045	0.20
3.93E-4	6.53E-4	27.8	153.2	0.1057	0.1058	0.50
6.15E-4	8.26E-4	30.2	106.7	0.1051	0.1052	0.59*
**	9.51E-4	**	95.76	**	0.1064	0.70*
	1.18E-3		82.43		0.1065	0.80*
	1.60E-3		76.13		0.1056	0.90*
	1.56E-3		82.55		0.1061	0.92*
for 7 grid levels and 641 nodes on finest grid						
1.07E-5	3.27E-5	321.05	1037.2	0.0424	0.0783	0.02
2.72E-5	7.90E-5	147.85	625.42	0.0783	0.0783	0.05
5.73E-5	1.48E-4	83.21	496.00	0.0785	0.0784	0.10
1.22E-4	2.66E-4	50.65	280.43	0.0787	0.0787	0.20
3.90E-4	5.94E-4	30.24	176.99	0.0797	0.0798	0.50
6.30E-4	7.88E-4	33.53	146.31	0.0790	0.0791	0.58*

* error in location of front, front lags

** numerical instabilities resulting in math overflows

Table 5.5.4: Test Problem #5, Results for Time = 15 days

where $\log(\epsilon_x) = -8$ and $\log(toler) = -5$

AG residual $\|F - Lu\|$	TRID residual $\|F - Lu\|$	AG CPU time sec	TRID CPU time sec	AG mass gain cm H_2O	TRID mass gain cm H_2O	Δt day
for 6 grid levels and 481 nodes on finest grid						
7.02E-6	1.51E-4	297.40	1265.8	0.109	0.109	0.10
1.49E-5	2.65E-4	202.30	721.58	0.109	0.109	0.20
2.37E-5	9.04E-4	145.47	511.39	0.111	0.111	0.50
3.83E-5	4.50E-4	153.31	691.82	0.111	0.084	0.57*
**	5.80E-4	**	434.96	**	0.111	0.60*
	6.49E-4		448.88		0.112	0.70*
	1.56E-3		82.55		0.106	0.80*
for 7 grid levels and 641 nodes on finest grid						
8.83E-6	1.35E-4	391.22	1552.7	0.082	0.082	0.10
1.78E-5	2.36E-4	246.01	1075.4	0.082	0.082	0.20
2.57E-5	4.57E-4	167.51	717.07	0.083	0.083	0.50
4.40E-5	5.06E-4	173.80	707.37	0.082	0.082	0.58*

* error in location of front, front lags

** numerical instabilities resulting in math overflows

Table 5.5.5: Test Problem #5, Results for Time = 35 days

where $\log(\epsilon_x) = -8$ and $\log(toler) = -5$

AG residual $\|F - Lu\|$	TRID residual $\|F - Lu\|$	AG CPU time sec	TRID CPU time sec	AG mass gain cm H_2O	TRID mass gain cm H_2O	Δt day
for 6 grid levels and 481 nodes on finest grid						
1.11E-5	1.50E-4	1021.1	2242.1	0.109	0.109	0.10
2.47E-5	2.66E-4	656.53	1705.6	0.109	0.109	0.20
1.13E-4	4.57E-4	535.42	1711.3	0.110	0.083	0.50
6.11E-5	5.66E-4	575.53	1277.5	0.111	0.111	0.57*
**	2.31E-3	**	1439.4	**	0.111	0.60*
for 7 grid levels and 641 nodes on finest grid						
8.12E-5	2.36E-4	899.81	2535.3	0.082	0.082	0.20
1.84E-4	4.57E-4	643.53	1721.8	0.083	0.083	0.50
9.23E-5	5.06E-4	711.81	2480.8	0.084	0.083	0.58*

* error in location of front, front lags

** numerical instabilities resulting in math overflows

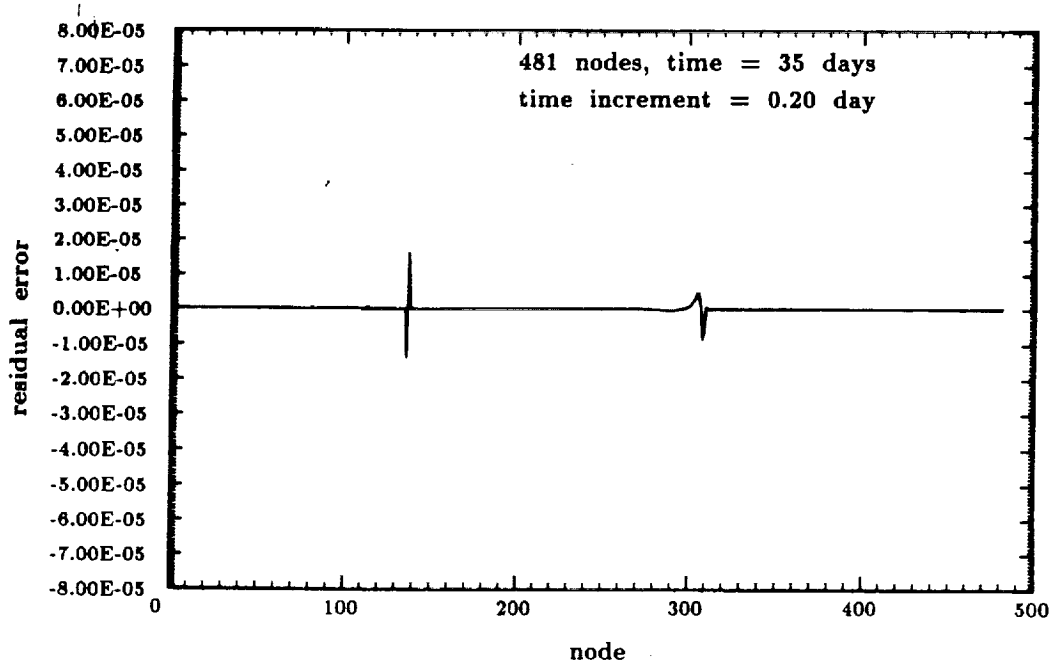

Figure 5.5.3: Adaptive Grid Residual Error, Case 1

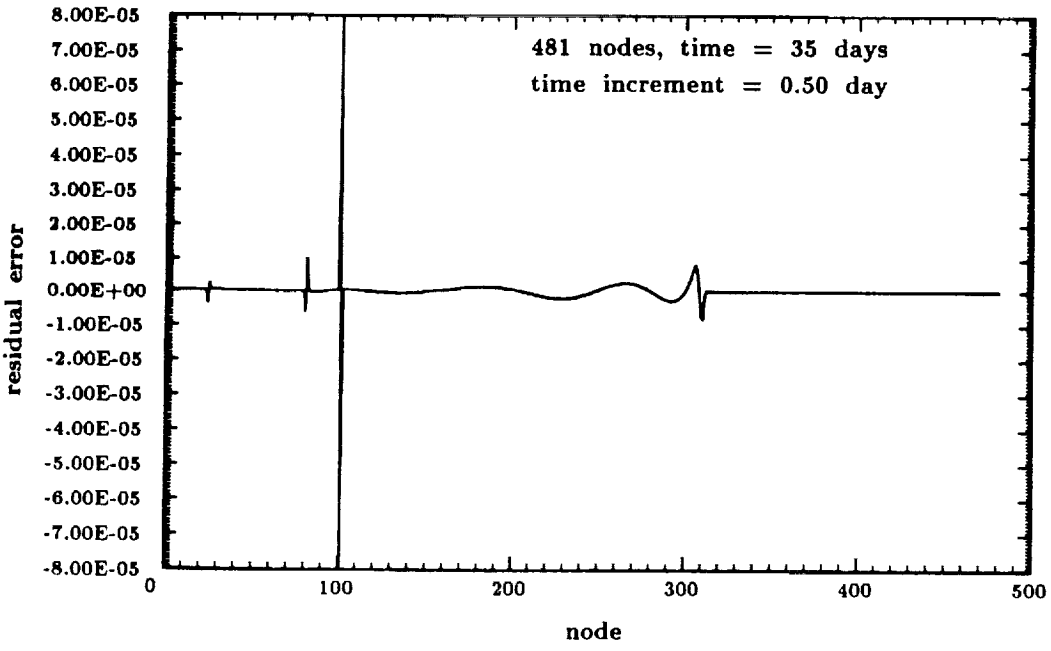

Figure 5.5.4: Adaptive Grid Residual Error, Case 2

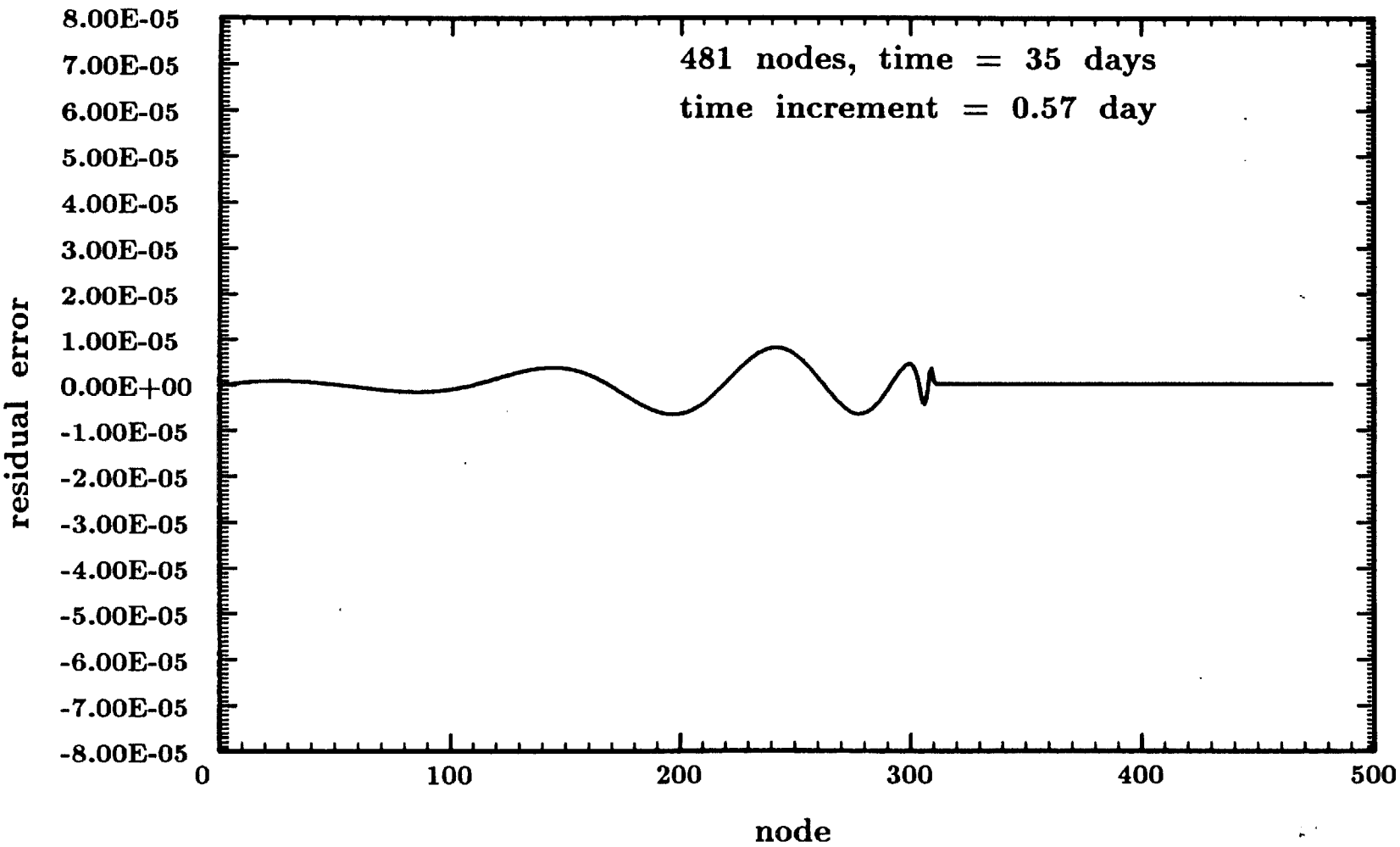

Figure 5.5.5: Adaptive Grid Residual Error, Case 3

For the adaptive grid method, using time steps larger than 0.58 day results in math overflows. For the tridiagonal scheme, the overflows occur at much larger time increments. Yet for both algorithms, the computed location of the front begins to lag behind its actual location for large time steps which do not induce math overflows. The lagging front becomes apparent for time increments larger than 0.50 day for both algorithms.

As with the previous problem, the size of the time increment is important. For this problem, time increments larger than 0.50 day will either yield solutions in which the front lags behind its actual location or produce math overflows. Time steps smaller than 0.50 day will generally result in solutions with a well defined and properly located front. While the use of smaller and smaller time increments will give an increasingly accurate solution, it is not without cost. A reduction in the size of the time step can significantly increase the computation time required to solve the problem, as is shown in Figures 5.5.6, 5.5.7, and 5.5.8 .

The tridiagonal method used to solve the Richards' equation used the adaptive grid residual norm, found from the finest grid level at the final time step, as its convergence criteria. As before, the tridiagonal program was modified such that the adaptive grid residual norm from each time step is used as the convergence criteria for the corresponding time increments. This change resulted in a slight improvement in the performance of the direct solution

method (see Table 5.5.6), with the tridiagonal program running about 1.1 to 1.2 times faster than before.

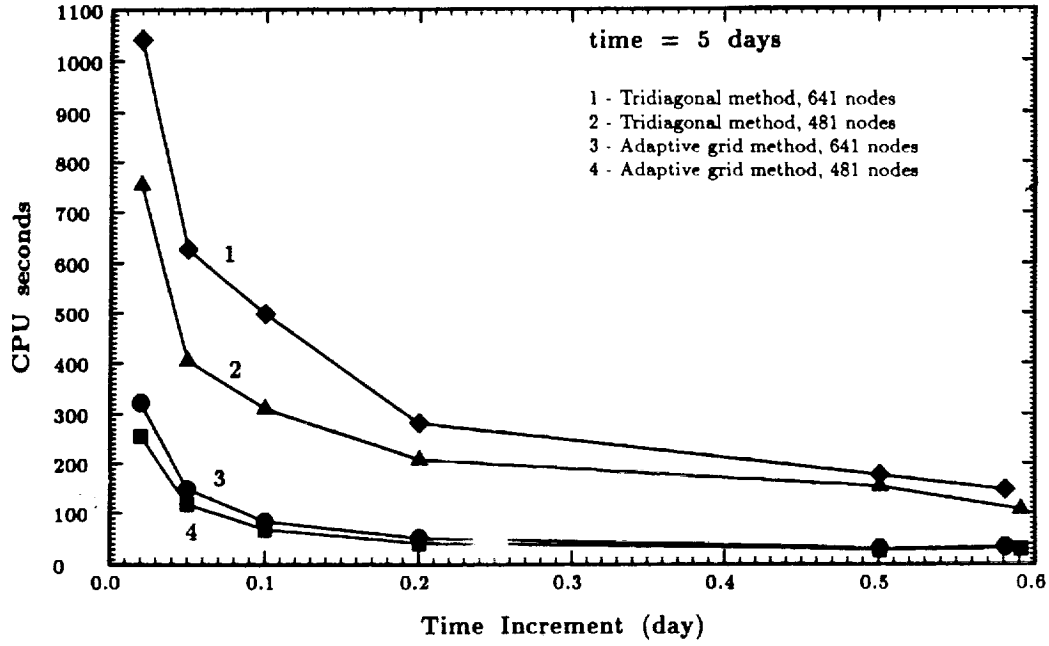

Figure 5.5.6: CPU Time vs. Time Step; Richards' Equation at Time = 5 days

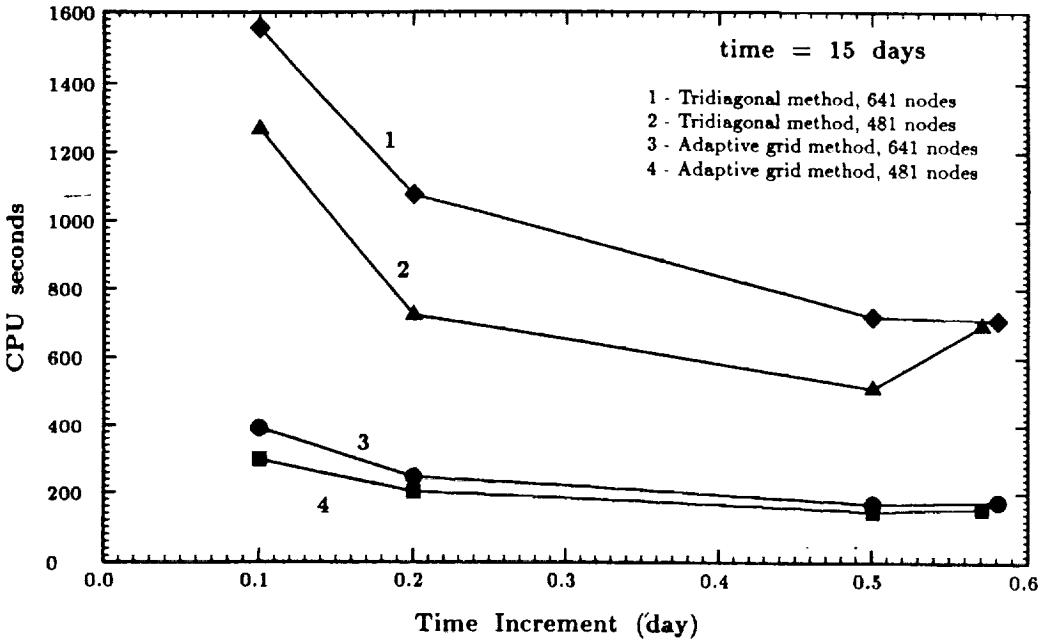

Figure 5.5.7: CPU Time vs. Time Step; Richards Equation at Time = 15 days

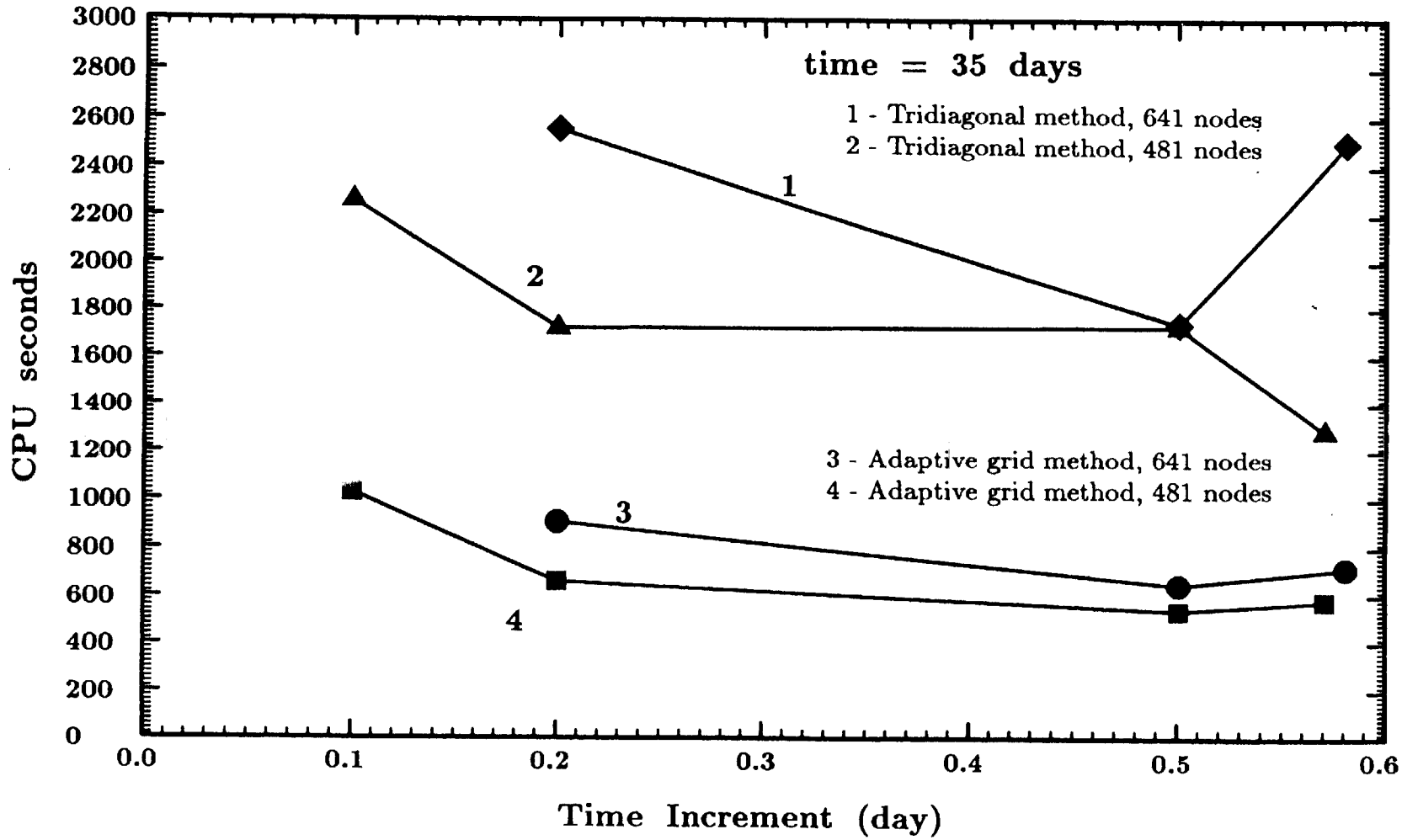

Figure 5.5.8: CPU Time vs. Time Step; Richards' Equation at Time = 35

Table 5.5.6: Results For the Alternate Tridiagonal Method

where $\log(\epsilon_r) = -8$ and $\log(toler) = -8$

Time day	Δt sec	former* mass gain cm H_2O	alt** mass gain cm H_2O	former* CPU time sec	alt** CPU time sec	speed up
5	0.5	.1057	.1055	153.2	124.3	1.232
5	0.2	.1045	.1038	206.9	204.2	1.013
5	0.1	.1042	.1032	310.0	250.0	1.240
15	0.5	.1110	.1103	511.1	434.6	1.176
35	0.2	.1090	.1083	1705.6	1552.1	1.099

* original tridiagonal solver; convergence criteria constant for all time increments

** alternate tridiagonal solver; convergence criteria varies with time steps

Chapter 6

Conclusions and Recommendations

6.1 Conclusions

The purpose of this project is to investigate the use of the adaptive grid and multigrid methods for the fast and efficient solution of one-dimensional problems, and apply the most promising of the methods toward finding the solution of the nonlinear, one-dimensional transport problems.

The adaptive grid and multigrid programs were first applied to the solution of linear one-dimensional problems. Test problem #1, an ordinary differential equation, is designed such that it contains both low and high frequency terms in its solution. In solving this problem, the tridiagonal and multigrid methods easily resolve both the high and low frequency terms, while the Gauss-Seidel method had to use a very large number of iterations to resolve the low frequency terms. For the transient cases (test problems #2 and #3), the performance of the multigrid method improves as steady state is approached. For these problems, the adaptive grid and multigrid programs are significantly faster than the Gauss-Seidel method, with the adaptive grid scheme running about 1.5 to 4 times faster than the multigrid method. While the adaptive grid algorithm is the fastest of the iterative methods, it is still about 3 times slower than the tridiagonal method.

In order to solve the nonlinear problems, Picard iterations are incorporated into the adaptive grid and tridiagonal programs. These two algorithms were used in solving the Burgers' equation. Both methods obtained solutions of similar accuracy, with the adaptive grid scheme finding the solution 3.26 times faster than the tridiagonal method. In solving the Richards' equation, the tridiagonal scheme has problems eliminating low frequency errors, and so is unable to achieve the accuracy obtained by the adaptive grid method. For this problem, the adaptive grid program is approximately 2 to 4.7 times faster than the tridiagonal scheme. For both nonlinear problems, the largest time increment yielding a satisfactory solution is the same for the two algorithms.

The performance of the adaptive grid and multigrid algorithms generally improved as the number of grid levels were increased. The adaptive grid method shows lots of promise toward the solution of nonlinear one-dimensional problems, despite being outperformed by the tridiagonal method when solving linear problems. For the nonlinear equations solved, the adaptive grid scheme is about 3 times faster than the iterative tridiagonal method. The adaptive grid program can easily smooth out the low and high frequency errors present in the approximation, even when these errors pose a problem to the iterative tridiagonal scheme. Thus in some cases, such as for the Richards' equation, the adaptive grid algorithm computes a more accurate solution than does the tridiagonal method. With problems which have moving fronts penetrating

into undisturbed portions of the domain, the performance of the adaptive grid scheme degrades as the solution progresses in time, but still remains faster than the tridiagonal method.

In Summary, (see Table 6.1.1) for the linear problems presented in this work, the adaptive and multigrid programs performed about the same, with the adaptive grid scheme being slightly faster than the multigrid method. Both the adaptive grid and multigrid programs easily outperformed the Gauss-Seidel method, but were approximately three to four times slower than the direct solver (tridiagonal method). For the solution of the nonlinear problems, Burgers' equation and Richards' equation, the situation reversed itself with the adaptive grid program obtaining the solutions about three times faster than the tridiagonal method incorporating Picard iterations.

Table 6.1.1: Summary of Representative Results

Test Problem	Gauss-Seidel CPU sec	Multigrid CPU sec	Adaptive Grid CPU sec	Tridiagonal CPU sec
#1	48.41	0.08	**	0.02
#2	116.60	7.82	5.31	1.74
#3	**	4.47	3.87	1.19
#4	**	**	9.32	30.37
#5 481 nodes time=5 days	**	**	67.70	309.95
#5 641 nodes time=5 days#5	**	**	83.21	496.00
#5 481 nodes time=35 days	**	**	656.53	1705.60
#5 641 nodes time=35 days	**	**	899.81	2535.30

** indicates method not used

6.2 Recommendations

The adaptive grid method performs well in finding the solution to the nonlinear problems considered here. Particularly when it is applied to solving the water content formulation of the Richards' equation. Further consideration of this method (and its derivatives) for the solution of the Richards' equation and other nonlinear problems is suggested.

Certain alterations to the adaptive grid program should be considered in order to improve the efficiency and accuracy of the algorithm. Currently, the nonlinear adaptive grid algorithm uses a series of finer and finer grids to solve a problem, but, it does not allow for any cycling between the coarser and finer grids. So, when the relaxation sweeps on a finer grid level become inefficient (due to the presence of low frequency errors), the algorithm does not seek to go to a coarser grid on which these errors can be easily reduced. Therefore, it is recommended that the adaptive grid and multigrid methods be combined so as to allow for a cycling of the solution process between the coarse and fine grids.

In the current adaptive grid algorithm (as well as with the tridiagonal method employing Picard iterations) small errors in mass balance are present. Both smaller time increments and denser grids containing more nodes may be used to improve the mass balance and the accuracy of the solution, but at the added expense of an increase in computational work (work units) and

computation time, which can be very significant. The adaptive grid method may offer a way to improve the mass balance at a minimal cost. Part of the problem lies with the maintenance of a proper mass balance in the approximation used on the various adaptive grids. The adaptive grid method presented uses Dirichlet conditions for the boundaries of the adaptive subdomains. The one exception to this rule applies to the subdomain boundaries corresponding to the actual boundary of the problem; in this case the actual boundary condition (as posed by the problem of interest) is used. The use of the Dirichlet condition for subdomain boundaries neglects to consider any flux entering or leaving the subdomain. This increases errors in the mass balance and gives rise to the appearance of an error (a spike) in the residual at the subdomain boundary, which can trigger the creation of an additional subdomain at a finer grid level. As a corrective step, Neumann conditions (or Robins conditions), which specify the value of the flux at a boundary, may be useful at the subdomain boundaries. Doing this will improve the maintenance of a proper mass balance, and either reduce or eliminate the spike in the residual error. Elimination of the spike is important, as the spike may result in the creation of additional, perhaps unnecessary subdomains.

As the program proceeds to step through time, the final solution at each time step is used as an the initial estimate for the next time step. While this works rather well, it introduces errors into the initial approximation for

the new time step as a proper mass balance is not maintained from one time increment to the next. These errors may induce additional computational work. Providing the new time step with a more accurate estimate of the end time step solution (with a correct mass balance) may serve to enhance the performance of the algorithm. For problems with steep moving fronts (such as for the Burgers' and Richards' equations solved in this study), the velocity of the front may be calculated from previous time steps and used to provide a better estimate of the solution to the next time increment.

References

[1] Alcouffe, R. E., Achi Brandt, J. E. Dendy Jr., and J. W. Painter, The Multigrid Method for the Diffusion Equation with Strongly Discontinuous Coefficients, *Siam Journal on Scientific and Statistical Computing*, Vol. 2, Num. 4, pp. 430-454, 1981.

[2] Brandt, Achi, Multi-Level Adaptive Technique (MLAT) For Fast Numerical Solution to Boundary Value Problems, *Lecture Notes in Physics*, Springer-Verlag, New York, Vol. 18, pp. 82-89, 1973.

[3] Brandt, Achi, Multi-Level Adaptive Solutions to Boundary-Value Problems, *Mathematics of Computation*, Vol. 31, Num. 138, pp. 333-390, April 1977.

[4] Brandt, Achi, Multilevel Adaptive Techniques (MLAT) for Singular-Perturbation Problems, *Numerical Analysis of Singular Perturbation Problems*, pp. 53-142, Academic Press, London, 1979.

[5] Brandt, Achi, Guide to Multigrid Development, *Lecture Notes in Mathematics: Multi-Grid Methods*, W. Hackbusch and U. Trottenberg, editors, Springer-Verlag, New York, 1982.

[6] Carey, G. F., and A. Pandanami, Multigrid Solution of Convection-Diffusion Problems, paper presented at the Fourth Copper Mountain Conference on Multigrid Methods, April 1989.

[7] Hedstrom G. W., and G. H. Rodrique, Adaptive-grid Methods for Time-dependent Partial Differential Equations, *Lecture Notes in Mathematics: Multi-Grid Methods*, W. Hackbusch and U. Trottenberg, editors, Springer-Verlag, New York, 1982.

[8] Heroux, Michael A., and J. W. Thomas, TDFAC: A Composite Grid Method for Time Dependent Problems, *Proceedings of the Fourth Copper Mountain Conference on Multigrid Methods*, edited by Jan Mandel, et. al., SIAM, Philadelphia, Pa, pp. 273-285, 1989.

[9] Hills, R. G., I. Porro, D. B. Hudson, and P. J. Wierenga, Modeling One-Dimensional Infiltration Into Very Dry Soils, 1, Model Development and Evaluation, *Water Resour. Res.*, Vol. 25 Num. 6, pp. 1259-1269, June 1989a.

[10] Hills, R. G., I. Porro, D. B. Hudson, and P. J. Wierenga, Modeling One-Dimensional Infiltration Into Very Dry Soils, 2, Estimation of the Soil Water Parameters and Model Predictions, *Water Resour. Res.*, Vol. 25 Num. 6, pp. 1271-1282, June 1989b.

[11] Jesperson, D. C., Multigrid Methods for Partial Differential Equations, *Studies in Numerical Analysis*, Gene G. Golub, editor, MAA Studies in Mathematics, 24, Mathematical Association of America, Washington D. C., 1984.

[12] Lee, H. N., and R. E. Meyers, On Time Dependent Numerical Technique, *Comp. and Math. with Appls.*, Vol. 6, pp. 61-65, Pergammon Press Ltd., Great Britain, 1980.

[13] Wesseling, P., A Robust and Efficient Multigrid Method, *Lecture Notes in Mathematics: Multi-Grid Methods*, W. Hackbusch and U. Trottenberg, editors, Springer-Verlag, New York, 1982.